



Developer Guide

Amazon Cognito



Amazon Cognito: Developer Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

What is Amazon Cognito?	1
User pools	2
Identity pools	3
Features of Amazon Cognito	4
User pools	4
Identity pools	6
Amazon Cognito user pools and identity pools comparison	8
Getting started with Amazon Cognito	12
Regional availability	13
Pricing for Amazon Cognito	13
Terms and concepts	13
General	14
User pools	16
Identity pools	21
Getting started with AWS	22
Sign up for an AWS account	22
Create a user with administrative access	22
Getting started with user pools	24
Your first app and user pool	24
Other application options	26
React SPA example	27
Flutter mobile app example	30
Next steps	33
Add a social provider	34
Add a SAML IdP	42
Getting started with identity pools	45
Create an identity pool in Amazon Cognito	45
Set up an SDK	47
Integrate the identity providers	47
Get credentials	48
Additional getting started options	49
Integrating with apps	51
Authentication with AWS Amplify	52
Creating a user interface (UI) with Amplify	53

Authentication with AWS SDKs	54
How authentication works	55
Managed login authentication	55
SDK authentication	59
Third-party identity provider authentication	62
Identity pool authentication	65
Working with AWS SDKs	68
Authorization with Amazon Verified Permissions	69
API authorization with Verified Permissions	71
Example policy for an Amazon Cognito user	74
Code examples	77
Amazon Cognito Identity	79
Basics	79
Scenarios	101
Amazon Cognito Identity Provider	103
Basics	114
Scenarios	247
Amazon Cognito Sync	397
Basics	398
Multi-tenancy best practices	401
Per-tenant user pools	403
Per-tenant app clients	405
Per-tenant user pool groups	407
Per-tenant custom attributes	409
Per-tenant custom scopes	411
Example resource	414
Multi-tenancy security recommendations	415
Common Amazon Cognito scenarios	416
Authenticate with a user pool	416
Access server-side resources	417
Access resources with API Gateway and Lambda	418
Access AWS services with a user pool and an identity pool	418
Authenticate with a third party and access AWS services with an identity pool	419
Access AWS AppSync resources with Amazon Cognito	420
Amazon Cognito user pools	422
Features	423

Sign-up	423
Sign-in	424
Managed login	425
Security	425
Custom user experience	426
Monitoring and analytics	426
Amazon Cognito identity pools integration	427
User pool feature plans	427
Select a feature plan	429
Features by plan	430
Essentials plan features	432
Plus plan features	436
Turn off ineligible features	439
Authentication	440
Implement authentication flows	441
Things to know	443
Authentication flow example	446
Managed login authentication	449
SDK authentication	452
Authentication flows	455
SDK authorization models	476
Application resources	490
Third-party IdP sign-in	492
How federated sign-in works in Amazon Cognito user pools	492
The responsibilities of an app as a service provider with Amazon Cognito	494
Things to know about Amazon Cognito user pools third-party sign-in	494
Identity providers	495
Social identity providers	501
SAML providers	509
OIDC providers	538
Mapping IdP attributes	548
Linking federated users	555
Managed login	558
Managed login localization	560
Setting up managed login with AWS Amplify	561
Setting up managed login with the Amazon Cognito console	562

Viewing your sign-in page	562
Customizing your authentication pages	564
Things to know about managed login and the hosted UI	564
Configuring a domain	567
Branding and customization	580
Using Lambda triggers	599
Important considerations	602
Adding a user pool trigger	604
User pool Lambda trigger event	605
User pool Lambda trigger common parameters	606
Lambda trigger sources by event	607
Lambda trigger sources by function	613
Pre sign-up Lambda trigger	616
Post confirmation Lambda trigger	626
Pre authentication Lambda trigger	630
Post authentication Lambda trigger	635
Challenge Lambda triggers	639
Pre token generation Lambda trigger	655
Migrate user Lambda trigger	674
Custom message Lambda trigger	680
Custom sender Lambda triggers	687
Managing users	705
Allowing user sign-up	706
Signing up and confirming user accounts	709
Creating users as administrator	735
Adding groups to a user pool	741
Managing and searching for users	744
Passwords	749
Importing users into a user pool	752
Attributes	770
User pool tokens	784
ID tokens	786
Access tokens	790
Refresh tokens	794
Revoking tokens	796
Verifying a JSON Web Token	798

Managing user pool token expiration and caching	803
Accessing resources after sign-in	807
Accessing resources with Verified Permissions	417
Accessing API Gateway resources	810
Accessing AWS resources using an identity pool	811
Additional features	816
Updating a user pool and app client	817
App clients	821
Working with devices	830
Access control with resource servers	836
Using Amazon Pinpoint analytics	844
Email settings	850
SMS message settings	863
Using security features	872
Adding MFA	874
Threat protection	888
AWS WAF Web ACLs	918
Case sensitivity	922
Deletion protection	923
Managing user disclosure	925
User pool endpoints reference	931
Managed login endpoints	932
Federation endpoints	939
OAuth 2.0 grants	965
Using PKCE	966
Managed login and federation error responses	968
Amazon Cognito identity pools	971
Configuring identity pools	973
Create an identity pool	974
User IAM roles	975
Authenticated and unauthenticated identities	976
Activate or deactivate guest access	976
Change the role associated with an identity type	977
Edit identity providers	978
Delete an identity pool	979
Delete an identity from an identity pool	980

Using Amazon Cognito Sync with identity pools	980
Identity pools authentication flow	983
IAM roles	993
Set up a trust policy	994
Access policies	997
Role trust and permissions	1008
Security best practices	1009
IAM configuration best practices	1010
Identity pool configuration best practices	1012
Using attributes for access control	1013
Using attributes for access control with Amazon Cognito identity pools	1014
Using attributes for access control policy example	1016
Turn off attributes for access control	1018
Default provider mappings	1018
Using role-based access control	1020
Creating roles for role mapping	1021
Granting pass-role permission	1021
Using tokens to assign roles to users	1022
Using rule-based mapping to assign roles to users	1023
Token claims to use in rule-based mapping	1025
Best practices for role-based access control	1026
Getting credentials	1027
Using credentials	1034
Third-party identity providers	1037
Facebook	1037
Login with Amazon	1046
Google	1051
Sign in with Apple	1063
Open ID Connect providers	1070
SAML identity providers	1073
Developer-authenticated identities	1077
Understanding the authentication flow	1077
Define a developer provider name and associate it with an identity pool	1078
Implement an identity provider	1078
Updating the logins map (Android and iOS only)	1087
Getting a token (server side)	1088

Connect to an existing social identity	1089
Supporting transition between providers	1090
Switching identities	1094
Android	1094
iOS - objective-C	1094
iOS - swift	1095
JavaScript	1095
Unity	1096
Xamarin	1097
Amazon Cognito Sync	1098
Getting started with Amazon Cognito Sync	1098
Set up an identity pool in Amazon Cognito	1099
Store and sync data	1099
Synchronizing data across clients	1099
Initializing the Amazon Cognito Sync client	1100
Understanding datasets	1102
Reading and writing data in datasets	1104
Synchronizing local data with the sync store	1106
Handling event callbacks	1109
Android	1110
iOS - Objective-C	1112
iOS - Swift	1115
JavaScript	1119
Unity	1121
Xamarin	1124
Implementing push synchronization	1127
Create an Amazon Simple Notification Service (Amazon SNS) app	1127
Enable push sync in the Amazon Cognito console	1127
Use push sync in your app: Android	1129
Use push sync in your app: iOS - Objective-C	1131
Use push sync in your app: iOS - Swift	1133
Implementing Amazon Cognito Sync streams	1136
Customizing workflows with Amazon Cognito Events	1139
Security	1144
Data protection	1145
Data encryption	1145

Identity and access management	1146
Audience	1147
Authenticating with identities	1147
Managing access using policies	1151
How Amazon Cognito works with IAM	1153
Identity-based policy examples	1162
Troubleshooting	1167
Using service-linked roles	1169
Logging and monitoring	1173
Monitoring costs	1174
Exporting user pool logs	1177
Monitoring quotas and usage	1187
CloudTrail logs	1199
Compliance validation	1227
Resilience	1228
Regional data considerations	1228
Infrastructure security	1229
Configuration and vulnerability analysis	1229
AWS managed policies	1230
Policy updates	1231
Tagging resources	1234
Supported resources	1234
Tag restrictions	1235
Managing tags with the console	1235
AWS CLI examples	1236
Assigning tags	1236
Viewing tags	1237
Removing tags	1237
Applying tags when you create resources	1238
API actions	1239
API actions for user pool tags	1239
API actions for identity pool tags	1239
Quotas	1240
Understanding API request rate quotas	1240
Quota categorization	1240
Amazon Cognito user pools API operations with special request rate handling	1241

Monthly active users	1242
Managing API request rate quotas	1243
Identify quota requirements	1243
Optimize request rates	1244
Track quota usage	1245
Track monthly active users (MAUs)	1246
Requesting a quota increase	1246
User pools request rate quotas	1247
Identity pools request rate quotas	1258
Quotas on resource number and size	1259
Document history	1266

What is Amazon Cognito?

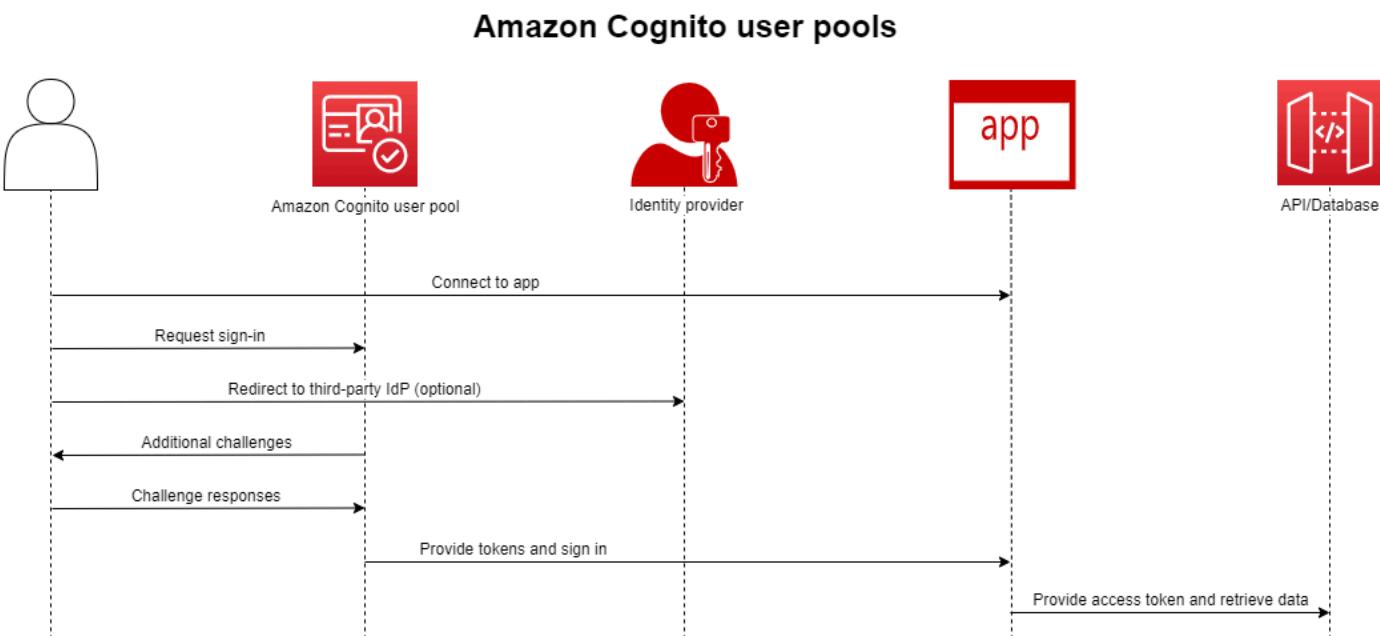
Amazon Cognito is an identity platform for web and mobile apps. It's a user directory, an authentication server, and an authorization service for OAuth 2.0 access tokens and AWS credentials. With Amazon Cognito, you can authenticate and authorize users from the built-in user directory, from your enterprise directory, and from consumer identity providers like Google and Facebook.

Topics

- [User pools](#)
- [Identity pools](#)
- [Features of Amazon Cognito](#)
- [Amazon Cognito user pools and identity pools comparison](#)
- [Getting started with Amazon Cognito](#)
- [Regional availability](#)
- [Pricing for Amazon Cognito](#)
- [Common Amazon Cognito terms and concepts](#)
- [Getting started with AWS](#)

The two components that follow make up Amazon Cognito. They operate independently or in tandem, based on your access needs for your users.

User pools

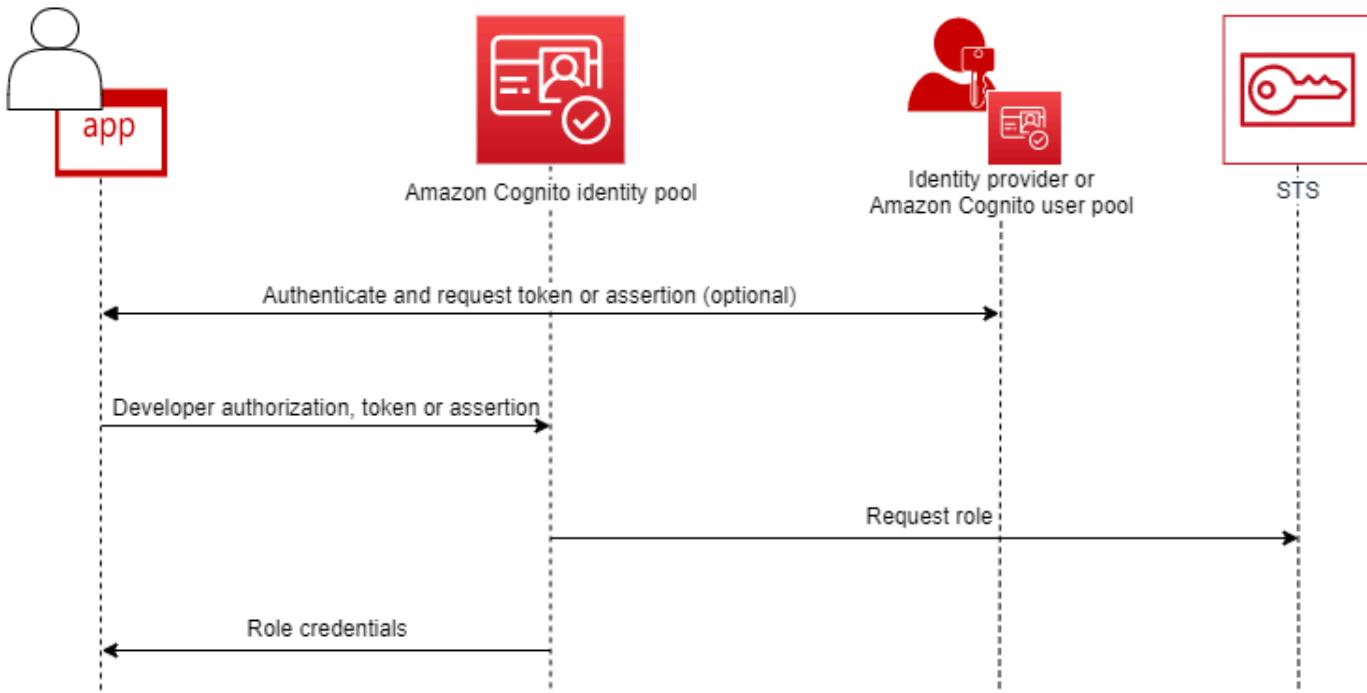


Create a user pool when you want to authenticate and authorize users to your app or API. User pools are a user directory with both self-service and administrator-driven user creation, management, and authentication. Your user pool can be an independent directory and OIDC identity provider (IdP), and an intermediate service provider (SP) to third-party providers of workforce and customer identities. You can provide single sign-on (SSO) in your app for your organization's workforce identities in SAML 2.0 and OIDC IdPs with user pools. You can also provide SSO in your app for your organization's customer identities in the public OAuth 2.0 identity stores Amazon, Google, Apple and Facebook. For more information about customer identity and access management (CIAM), see [What is CIAM?](#).

User pools don't require integration with an identity pool. From a user pool, you can issue authenticated JSON web tokens (JWTs) directly to an app, a web server, or an API.

Identity pools

Amazon Cognito federated identities (identity pools)



Set up an Amazon Cognito identity pool when you want to authorize authenticated or anonymous users to access your AWS resources. An identity pool issues AWS credentials for your app to serve resources to users. You can authenticate users with a trusted identity provider, like a user pool or a SAML 2.0 service. It can also optionally issue credentials for guest users. Identity pools use both role-based and attribute-based access control to manage your users' authorization to access your AWS resources.

Identity pools don't require integration with a user pool. An identity pool can accept authenticated claims directly from both workforce and consumer identity providers.

An Amazon Cognito user pool and identity pool used together

In the diagram that begins this topic, you use Amazon Cognito to authenticate your user and then grant them access to an AWS service.

1. Your app user signs in through a user pool and receives OAuth 2.0 tokens.

2. Your app exchanges a user pool token with an identity pool for temporary AWS credentials that you can use with AWS APIs and the AWS Command Line Interface (AWS CLI).
3. Your app assigns the credentials session to your user, and delivers authorized access to AWS services like Amazon S3 and Amazon DynamoDB.

For more examples that use identity pools and user pools, see [Common Amazon Cognito scenarios](#).

In Amazon Cognito, the *security of the cloud* obligation of the [shared responsibility model](#) is compliant with SOC 1-3, PCI DSS, ISO 27001, and is HIPAA-BAA eligible. You can design your *security in the cloud* in Amazon Cognito to be compliant with SOC1-3, ISO 27001, and HIPAA-BAA, but not PCI DSS. For more information, see [AWS services in scope](#). See also [Regional data considerations](#).

Features of Amazon Cognito

User pools

An Amazon Cognito user pool is a user directory. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito, or federate through a third-party IdP. Federated and local users have a user profile in your user pool.

Local users are those who signed up or you created directly in your user pool. You can manage and customize these user profiles in the AWS Management Console, an AWS SDK, or the AWS Command Line Interface (AWS CLI).

Amazon Cognito user pools accept tokens and assertions from third-party IdPs, and collect the user attributes into a JWT that it issues to your app. You can standardize your app on one set of JWTs while Amazon Cognito handles the interactions with IdPs, mapping their claims to a central token format.

An Amazon Cognito user pool can be a standalone IdP. Amazon Cognito draws from the OpenID Connect (OIDC) standard to generate JWTs for authentication and authorization. When you sign in local users, your user pool is authoritative for those users. You have access to the following features when you authenticate local users.

- Implement your own web front-end that calls the Amazon Cognito user pools API to authenticate, authorize, and manage your users.

- Set up multi-factor authentication (MFA) for your users. Amazon Cognito supports time-based one-time password (TOTP) and SMS message MFA.
- Secure against access from user accounts that are under malicious control.
- Create your own custom multi-step authentication flows.
- Look up users in another directory and migrate them to Amazon Cognito.

An Amazon Cognito user pool can also fulfill a dual role as a service provider (SP) to your IdPs, and an IdP to your app. Amazon Cognito user pools can connect to consumer IdPs like Facebook and Google, or workforce IdPs like Okta and Active Directory Federation Services (ADFS).

With the OAuth 2.0 and OpenID Connect (OIDC) tokens that an Amazon Cognito user pool issues, you can

- Accept an ID token in your app that authenticates a user, and provides the information that you need to set up the user's profile
- Accept an access token in your API with the OIDC scopes that authorize your users' API calls.
- Retrieve AWS credentials from an Amazon Cognito identity pool.

Features of Amazon Cognito user pools

Feature	Description
OIDC IdP	Issue ID tokens to authenticate users
Authorization server	Issue access tokens to authorize user access to APIs
SAML 2.0 SP	Transform SAML assertions into ID and access tokens
OIDC SP	Transform OIDC tokens into ID and access tokens
OAuth 2.0 SP	Transform ID tokens from Apple, Facebook, Amazon, or Google to your own ID and access tokens

Authentication frontend service	Sign up, manage, and authenticate users with managed login
API support for your own UI	Create, manage and authenticate users through API requests in supported AWS SDKs ¹
MFA	Use SMS messages, TOTPs, or your user's device as an additional authentication factor ¹
Security monitoring & response	Secure against malicious activity and insecure passwords ¹
Customize authentication flows	Build your own authentication mechanism, or add custom steps to existing flows ¹
Groups	Create logical groupings of users, and a hierarchy of IAM role claims when you pass tokens to identity pools
Customize ID tokens	Customize your ID tokens with new, modified, and suppressed claims
Customize user attributes	Assign values to user attributes and add your own custom attributes

¹ Feature is only available to local users.

For more information about user pools, see [Getting started with user pools](#) and the [Amazon Cognito user pools API reference](#).

Identity pools

An identity pool is a collection of unique identifiers, or identities, that you assign to your users or guests and authorize to receive temporary AWS credentials. When you present proof of authentication to an identity pool in the form of the trusted claims from a SAML 2.0, OpenID Connect (OIDC), or OAuth 2.0 social identity provider (IdP), you associate your user with an identity in the identity pool. The token that your identity pool creates for the identity can retrieve temporary session credentials from AWS Security Token Service (AWS STS).

To complement authenticated identities, you can also configure an identity pool to authorize AWS access without IdP authentication. You can offer your own custom proof of authentication, or no authentication. You can grant temporary AWS credentials to any app user who requests them, with [unauthenticated identities](#). Identity pools also accept claims and issue credentials based on your own custom schema, with [developer-authenticated identities](#).

With Amazon Cognito identity pools, you have two ways to integrate with IAM policies in your AWS account. You can use these two features together or individually.

Role-based access control

When your user passes claims to your identity pool, Amazon Cognito chooses the IAM role that it requests. To customize the role's permissions to your needs, you apply IAM policies to each role. For example, if your user demonstrates that they are in the marketing department, they receive credentials for a role with policies tailored to marketing department access needs. Amazon Cognito can request a default role, a role based on rules that query your user's claims, or a role based on your user's group membership in a user pool. You can also configure the role trust policy so that IAM trusts only your identity pool to generate temporary sessions.

Attributes for access control

Your identity pool reads attributes from your user's claims, and maps them to principal tags in your user's temporary session. You can then configure your IAM resource-based policies to allow or deny access to resources based on IAM principals that carry the session tags from your identity pool. For example, if your user demonstrates that they are in the marketing department, AWS STS tags their session `Department: marketing`. Your Amazon S3 bucket permits read operations based on an [aws:PrincipalTag](#) condition that requires a value of `marketing` for the `Department` tag.

Features of Amazon Cognito identity pools

Feature	Description
Amazon Cognito user pool SP	Exchange an ID token from your user pool for web identity credentials from AWS STS
SAML 2.0 SP	Exchange SAML assertions for web identity credentials from AWS STS
OIDC SP	Exchange OIDC tokens for web identity credentials from AWS STS

OAuth 2.0 SP	Exchange OAuth tokens from Amazon, Facebook, Google, Apple, and Twitter for web identity credentials from AWS STS
Custom SP	With AWS credentials, exchange claims in any format for web identity credentials from AWS STS
Unauthenticated access	Issue limited-access web identity credentials from AWS STS without authentication
Role-based access control	Choose an IAM role for your authenticated user based on their claims, and configure your roles to only be assumed in the context of your identity pool
Attribute-based access control	Convert claims into principal tags for your AWS STS temporary session, and use IAM policies to filter resource access based on principal tags

For more information about identity pools, see [Getting started with Amazon Cognito identity pools](#) and the [Amazon Cognito identity pools API reference](#).

Amazon Cognito user pools and identity pools comparison

Feature	Description	User pools	Identity pools
OIDC IdP	Issue OIDC ID tokens to authenticate app users	✓	
API authorization server	Issue access tokens to authorize user access to APIs, databases, and other resources	✓	

	that accept OAuth 2.0 authorization scopes	
IAM web identity authorization server	Generate tokens that you can exchange with AWS STS for temporary AWS credentials	✓
SAML 2.0 SP & OIDC IdP	Issue customized OIDC tokens based on claims from a SAML 2.0 IdP	✓
OIDC SP & OIDC IdP	Issue customized OIDC tokens based on claims from an OIDC IdP	✓
OAuth 2.0 SP & OIDC IdP	Issue customized OIDC tokens based on scopes from OAuth 2.0 social providers like Apple and Google	✓
SAML 2.0 SP & credentials broker	Issue temporary AWS credentials based on claims from a SAML 2.0 IdP	✓
OIDC SP & credentials broker	Issue temporary AWS credentials based on claims from an OIDC IdP	✓

OAuth 2.0 SP & credentials broker	Issue temporary AWS credentials based on scopes from OAuth 2.0 social providers like Apple and Google	✓
Amazon Cognito user pool SP & credentials broker	Issue temporary AWS credentials based on OIDC claims from an Amazon Cognito user pool	✓
Custom SP & credentials broker	Issue temporary AWS credentials based on developer IAM authorization	✓
Authentication frontend service	Sign up, manage, and authenticate users with managed login	✓
API support for your own authentication UI	Create, manage and authenticate users through API requests in supported AWS SDKs ¹	✓
MFA	Use SMS messages, TOTPs, or your user's device as an additional authentication factor ¹	✓
Security monitoring & response	Protect against malicious activity and insecure passwords ¹	✓

Customize authentication flows	Build your own authentication mechanism, or add custom steps to existing flows ¹	✓
Groups	Create logical groupings of users, and a hierarchy of IAM role claims when you pass tokens to identity pools	✓
Customize ID tokens	Customize your ID tokens with new, modified, and suppressed claims	✓
AWS WAF web ACLs	Monitor and control requests to your authentication environment with AWS WAF	✓
Customize user attributes	Assign values to user attributes and add your own custom attributes	✓
Unauthenticated access	Issue limited-access web identity credentials from AWS STS without authentication	✓

Role-based access control	Choose an IAM role for your authenticated user based on their claims, and configure your roles to only be assumed in the context of your identity pool	✓
Attribute-based access control	Transform user claims into principal tags for your AWS STS temporary session, and use IAM policies to filter resource access based on principal tags	✓

¹ Feature is only available to local users.

Getting started with Amazon Cognito

For example user pool applications, see [Getting started with user pools](#).

For an introduction to identity pools, see [Getting started with Amazon Cognito identity pools](#).

For links to guided setup experiences with user pools and identity pools, see [Guided setup options for Amazon Cognito](#).

For videos, articles, documentation, and more sample applications, see [Amazon Cognito developer resources](#).

To use Amazon Cognito, you need an AWS account. For more information, see [Getting started with AWS](#).

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.

To see if Amazon Cognito is currently available in any AWS Region, see [AWS Services by Region](#).

To learn about regional API service endpoints, 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](#).

Common Amazon Cognito terms and concepts

Amazon Cognito provides credentials for web and mobile apps. It draws from and builds on terms that are common in *identity and access management*. Many guides to universal identity and access terms are available. Some examples are:

- [Terminology](#) in the IDPro Body of Knowledge
- [AWS Identity Services](#)
- [Glossary](#) from NIST CSRC

The following lists describe terms that are unique to Amazon Cognito or have a specific context in Amazon Cognito.

Topics

- [General](#)
- [User pools](#)
- [Identity pools](#)

General

The terms in this list aren't specific to Amazon Cognito and are widely recognized among identity and access management practitioners. The following isn't an exhaustive list of terms, but a guide to their specific Amazon Cognito context in this guide.

Access token

A JSON web token (JWT) that contains information about an entity's [authorization](#) to access information systems.

App, application

Typically, a mobile application. In this guide, *app* is often a shorthand for a web application or mobile app that connects to Amazon Cognito.

Attribute-based access control (ABAC)

A model where an app determines access to resources based on the properties of a user, like their job title or department. Amazon Cognito tools to enforce ABAC include ID tokens in user pools and [principal tags](#) in identity pools.

Authentication

The process of establishing an authentic identity for the purpose of access to an information system. Amazon Cognito accepts proof of authentication from third-party identity providers, and also serves as a provider of authentication to software applications.

Authorization

The process of granting permissions to a resource. User pool [access tokens](#) contain information that applications can use to permit users and systems to access resources.

Authorization server

An OAuth or OpenID Connect (OIDC) system that generates [JSON web tokens](#). The Amazon Cognito user pools [managed authorization server](#) is the authorization-server component of the two authentication and authorization methods in user pools. User pools also support API challenge-response flows in [SDK authentication](#).

Confidential app, server-side app

An application that users connect to remotely, with code on an application server and access to secrets. This is typically a web application.

Identity provider (IdP)

A service that stores and verifies user identities. Amazon Cognito can request authentication from [external providers](#) and be an IdP to apps.

JSON web token (JWT)

A JSON-formatted document that contains claims about an authenticated user. ID tokens authenticate users, access tokens authorize users, and refresh tokens update credentials.

Amazon Cognito receives tokens from [external providers](#) and issues tokens to apps or AWS STS.

Machine-to-machine (M2M) authorization

The process of authorizing requests to API endpoints for non-user-interactive machine entities, like a webserver application tier. User pools serve M2M authorization in client-credentials grants with OAuth 2.0 scopes in [access tokens](#).

Multi-factor authentication (MFA)

The requirement that users provide additional authentication after providing their username and password. Amazon Cognito user pools have MFA features for [local users](#).

OAuth 2.0 (social) provider

An IdP to a user pool or identity pool that provides [JWT](#) access and refresh tokens. Amazon Cognito user pools automate interactions with social providers after users authenticate.

OpenID Connect (OIDC) provider

An IdP to a user pool or identity pool that extends the [OAuth](#) specification to provide ID tokens. Amazon Cognito user pools automate interactions with OIDC providers after users authenticate.

Passkey, WebAuthn

A form of authentication where cryptographic keys, or passkeys, on a user's device provides their proof of authentication. Users verify that they are present with biometric or PIN code mechanisms in a hardware or software authenticator. Passkeys are phishing-resistant and bound to specific websites/apps, offering a secure passwordless experience. Amazon Cognito user pools support sign-in with passkeys.

Passwordless

A form of authentication where a user doesn't have to enter a password. Methods of passwordless sign-in include one-time passwords (OTPs) sent to email addresses and phone numbers, and passkeys. Amazon Cognito user pools support sign-in with OTPs and passkeys.

Public app

An application that is self-contained on a device, with code stored locally and no access to secrets. This is typically a mobile app.

Resource server

An API with access control. Amazon Cognito user pools also use *resource server* to describe the component that defines the configuration for interacting with an API.

Role-based access control (RBAC)

A model that grants access based on a user's functional designation. Amazon Cognito identity pools implement RBAC with differentiation between IAM roles.

Service provider (SP), relying party (RP)

An application that relies on an IdP to assert that users are trustworthy. Amazon Cognito acts as an SP to external IdPs, and as an IdP to app-based SPs.

SAML provider

An IdP to a user pool or identity pool that generates digitally signed assertion documents that your user passes to Amazon Cognito.

Universally Unique Identifier (UUID)

A 128-bit label that is applied to an object. Amazon Cognito UUIDs are unique per user pool or identity pool, but don't conform to a specific UUID format.

User directory

A collection of users and their attributes that serves that information to other systems. Amazon Cognito user pools are user directories, and also tools for consolidation of users from external user directories.

User pools

When you see the terms in the following list in this guide, they refer to a specific feature or configuration of user pools.

Adaptive authentication

A feature of [advanced security](#) that detects potential malicious activity and applies additional security to [user profiles](#).

Advanced security features

An optional component that adds tools for user security.

App client

A component that defines the settings for a user pool as an IdP to one app.

Callback URL, redirect URI, return URL

A setting in an [app client](#) and a parameter in requests to the user pool's [authorization server](#).

The callback URL is the initial destination for authenticated users in your [app](#).

Choice-based authentication

A form of API authentication with users pools where each user has a set of choices for sign-in available to them. Their choices might include username and password with or without MFA, passkey sign-in, or passwordless sign-in with email or SMS message one-time passwords. Your application can shape the choice process for users by requesting a list of authentication options or by declaring a preferred option.

Compare with [client-based authentication](#).

Client-based authentication

A form of authentication with the user pools API and application back ends built with AWS SDKs. In declarative authentication, your application determines independently the login type that a user should perform and requests that type up front.

Compare with [choice-based authentication](#).

Compromised credentials

A feature of [advanced security](#) that detects user passwords that attackers might know, and applies additional security to [user profiles](#).

Confirmation

The process that determines that the prerequisites have been met to permit a new user to sign in. Confirmation is typically done through email address or phone number [verification](#).

Custom authentication

An extension of authentication processes with [Lambda triggers](#) that define additional user challenges and responses.

Device authentication

An authentication process that replaces [MFA](#) with sign-in that uses the ID of a trusted device.

Domain, user pool domain

A web domain that hosts your [managed login pages](#) in AWS. You can set up DNS in a domain that you own or use an identifying subdomain prefix in a domain that AWS owns.

Essentials plan

The [feature plan](#) with the latest developments in user pools. The Essentials plan doesn't include the automated-learning security features in the [Plus plan](#).

External provider, third-party provider

An IdP that has a trust relationship with a user pool. User pools serve as an intermediate entity between external providers and your application, managing authentication processes with SAML 2.0, OIDC, and social providers. User pools consolidate external-provider authentication outcomes into a single IdP so that your applications can process many users with a single OIDC relying-party library.

Feature plan

The group of features that you can select for a user pool. Feature plans have differing costs in your AWS bill. New user pools default to the [Essentials plan](#).

Current plans

- [Lite plan](#)
- [Essentials plan](#)
- [Plus plan](#)

Federated user, external user

A user in a user pool who was authenticated by an [external provider](#).

Hosted UI (classic), hosted UI pages

The early version of the authentication front end, relying party, and identity provider services on your user pool domain. The hosted UI has a basic set of features and a simplified look and feel. You can apply Hosted UI branding with the upload of a logo-image file and a file with a predetermined set of CSS styles. Compare to [managed login](#).

Lambda trigger

A function in AWS Lambda that a user pool can automatically invoke at key points in user authentication processes. You can use Lambda triggers to customize authentication outcomes.

Local user

A [user profile](#) in the user pool [user directory](#) that wasn't created by authentication with an [external provider](#).

Linked user

A user from an [external provider](#) whose identity is merged with a [local user](#).

Lite plan

The [feature plan](#) with the features that originally launched with user pools. The Lite plan doesn't include the new features in the [Essentials plan](#) or the automated-learning security features in the [Plus plan](#).

Managed authorization server, hosted UI authorization server, authorization server

A component of [managed login](#) that hosts services for interaction with IdPs and apps on your [user pool domain](#). The [hosted UI](#) differs from managed login in the user-interactive features it offers, but has the same authorization-server capabilities.

Managed login, managed login pages

A set of webpages on your [user pool domain](#) that host services for user authentication. These services include functions for operating as an [IdP](#), a [relying party](#) for third-party IdPs, and a server of a user-interactive authentication UI. When you set up a domain for your user pool, Amazon Cognito brings all managed login pages online.

Your application import OIDC libraries that invoke users' browsers and direct them to the managed login UI for sign-up, sign-in, password management, and other authentication operations. After authentication, the OIDC libraries can process the outcome of the authentication request.

Managed login authentication

Sign-in with the services on your [user pool domain](#), done with user-interactive browser pages or HTTPS API requests. Applications handle managed login authentication with OpenID Connect (OIDC) libraries. This process includes sign-in with [external providers](#), local-user sign-in with

interactive managed login pages, and [M2M authorization](#). Authentication with the classic [hosted UI](#) also fall under this term.

Compare to [AWS SDK authentication](#).

Plus plan

The [feature plan](#) with the latest developments and advanced security features in user pools.

SDK authentication, AWS SDK authentication

A set of authentication and authorization API operations that you can add to your application back end with an AWS SDK. This authentication model requires your own custom-built login mechanism. The API can sign in [local users](#) and [linked users](#).

Compare to [managed login authentication](#).

Threat protection

In user pools, threat protection refers to technologies that are designed to mitigate threats to your authentication and authorization mechanisms. Adaptive authentication, compromised-credentials detection, and IP address blocklists are under the category of threat protection.

Token customization

The outcome of a pre token generation [Lambda trigger](#) that modifies a user's ID or access token at runtime.

User pool, Amazon Cognito identity provider, cognito-**idp**, Amazon Cognito user pools

An AWS resource with authentication and authorization services for applications that work with OIDC IdPs.

Verification

The process of confirming that a user owns an email address or phone number. A user pool sends a code to a user who has entered a new email address or phone number. When they submit the code to Amazon Cognito, they verify their ownership of the message destination and can receive additional messages from the user pool. Also, see [confirmation](#).

User profile, user account

An entry for a user in the [user directory](#). All users, including those from third-party IdPs, have a profile in their user pool.

Identity pools

When you see the terms in the following list in this guide, they refer to a specific feature or configuration of identity pools.

Attributes for access control

An implementation of [attribute-based access control](#) in identity pools. Identity pools apply user attributes as tags to user credentials.

Basic (classic) authentication

An authentication process where you can customize the request for [user credentials](#).

Developer authenticated identities

An authentication process that authorizes identity pool [user credentials](#) with [developer credentials](#).

Developer credentials

The IAM API keys of an identity pool administrator.

Enhanced authentication

An authentication flow that selects an IAM role and applies principal tags according to the logic that you define in your identity pool.

Identity

A [UUID](#) that links an app user and their [user credentials](#) to their profile in an external [user directory](#) that has a trust relationship with an identity pool.

Identity pool, Amazon Cognito federated identities, Amazon Cognito identity, cognito-identity

An AWS resource with authentication and authorization services for applications that use [temporary AWS credentials](#).

Unauthenticated identity

A user who has not signed in with an identity pool IdP. You can permit users to generate limited user credentials for a single IAM role before they authentication.

User credentials

Temporary AWS API keys that users receive after identity pool authentication.

Getting started with AWS

Before you start working with Amazon Cognito, set yourself up with some required AWS resources. If you can already sign in to an AWS account, you can skip this section. Continue reading if you are looking for information about signing up and signing in with AWS credentials. After you have credentials with sufficient AWS Identity and Access Management (IAM) permissions, you can get started with [user pools](#) and [identity pools](#).

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

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.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

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**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Getting started with user pools

You have an application that requires authentication and access control. You can work within the OpenID Connect (OIDC) framework for single sign-on (SSO). Amazon Cognito has tools for handling the logic of authentication in the application back end with an AWS SDK, and for invoking a browser in your client to access a managed authorization server.

The Amazon Cognito console guides you through the creation of a user pool from the view of your preferred application framework. From there, you can continue on to add features like federated sign-in with external [social](#) or [SAML 2.0](#) identity providers (IdPs). The application models in the Amazon Cognito console lean on the addition of OIDC libraries to your project and invoking a browser.

As you work to expand your feature set and incorporate more components of Amazon Cognito, read the [Amazon Cognito user pools](#) chapter for full descriptions of everything you can do with user pools.

The examples in this chapter and in the Amazon Cognito console demonstrate a basic integration of application resources with Amazon Cognito user pools. Later, you can adjust your user pool to use more of the options that are available to you. Then you can update your application to adopt new features and interact with IdPs.

If you don't want to use the [managed login pages](#), you can create an application with custom-built authentication interfaces using an AWS SDK or AWS Amplify. Applications that you build in this way interact with the [user pools API](#) and are suitable only for authenticating [local users](#). Continue learning about this authentication model at [Other application options](#).

Topics

- [Create a new application in the Amazon Cognito console](#)
- [Other application options](#)
- [Add more features and security options to your user pool](#)

Create a new application in the Amazon Cognito console

User pools add authentication options to software applications. For the easiest getting-started experience, step into the Amazon Cognito console and follow the instructions there. The creation

process there guides you not only through setup of user pool resources, but through setting up the initial pieces of your application.

When you're ready to begin, navigate to the [Amazon Cognito console](#) and select the button to create a new user pool. The setup process will guide you through your configuration and programming-language options.

Additional resources for authentication concepts

- [Authentication with Amazon Cognito user pools](#)
- [Understanding API, OIDC, and managed login pages authentication](#)
- [How authentication works with Amazon Cognito](#)
- [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#)

To create Amazon Cognito resources for your application

1. Navigate to the [Amazon Cognito console](#).
2. Select **Create user pool** from the **User pools** menu, or select **Get started for free in less than five minutes**.
3. Under **Define your application**, choose the **Application type** that best fits the application scenario that you want to create authentication and authorization services for.
4. In **Name your application**, enter a descriptive name or proceed with the default name.
5. You must make some basic choices under **Configure options** that support settings that you can't change after you create your user pool.
 - a. Under **Options for sign-in identifiers**, tell us how you want to identify users when they sign in. You can prefer user-generated usernames, email addresses, or phone numbers. You can also allow a combination of multiple options. Amazon Cognito accepts the options that you configure here in the username field of [managed login](#) sign-in forms.
 - b. Under **Required attributes for sign-up**, tell us what user information you want to collect when users register for a new account. In managed login pages, Amazon Cognito presents prompts for all required attributes.

Options for sign-in identifiers influences your required attributes. **Username** requires email or phone attributes for each user so that they can receive a password-reset code in an email or SMS message. **Email** requires the email attribute, and **Phone number** requires the phone number attribute.

6. Under **Add a return URL**, enter a redirect path to your application for after users complete authentication. This location should be a route in your application that uses OpenID Connect (OIDC) libraries to process user-authentication outcomes.
7. Choose **Create your application**. Amazon Cognito creates a user pool and app client with default settings for your application type. You can configure additional options like [external identity providers](#) and [multi-factor authentication \(MFA\)](#) after you create your initial resources.
8. On the **Set up your application** page, you can immediately get code examples for your application. To explore your new user pool, scroll down and select **Go to overview**.
9. To add more applications in the same user pool, navigate to the **App clients** menu and add a new app client. This will repeat the process of application-focused creation, but only add a new app client to the existing user pool.

After you create a user pool and one or more app clients with this process, you can start testing authentication operations with managed login. These quick-start options are open to public self sign-up. We recommend that you create a testing environment with the console process, then move your finalized design to production. Spend time familiarizing yourself with the capabilities of Amazon Cognito. Then, to move to production workloads, craft custom configurations and deploy them with automation tools like AWS CloudFormation and the AWS Cloud Development Kit (AWS CDK).

Other application options

You might have an existing application UI that you want to integrate with Amazon Cognito authentication. You might even have your own existing authentication pages with a less-functional directory setup than Amazon Cognito user pools. You can add or replace an authentication component to an application of this type with Amazon Cognito integrations in AWS SDKs for a variety of programming languages. Some examples follow.

If you create a user pool for this purpose in the Amazon Cognito console, it might not be necessary to have a [user pool domain](#) that hosts interactive sign-in pages and OpenID Connect (OIDC) services. The process of user pool creation in the console automatically generates a domain for you. You can delete this domain from the **Domain** tab of your user pool. Other options include programmatic creation of Amazon Cognito resources for your application with API requests in AWS SDKs and with the automated-setup options in the AWS Amplify CLI. For more information, see [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#).

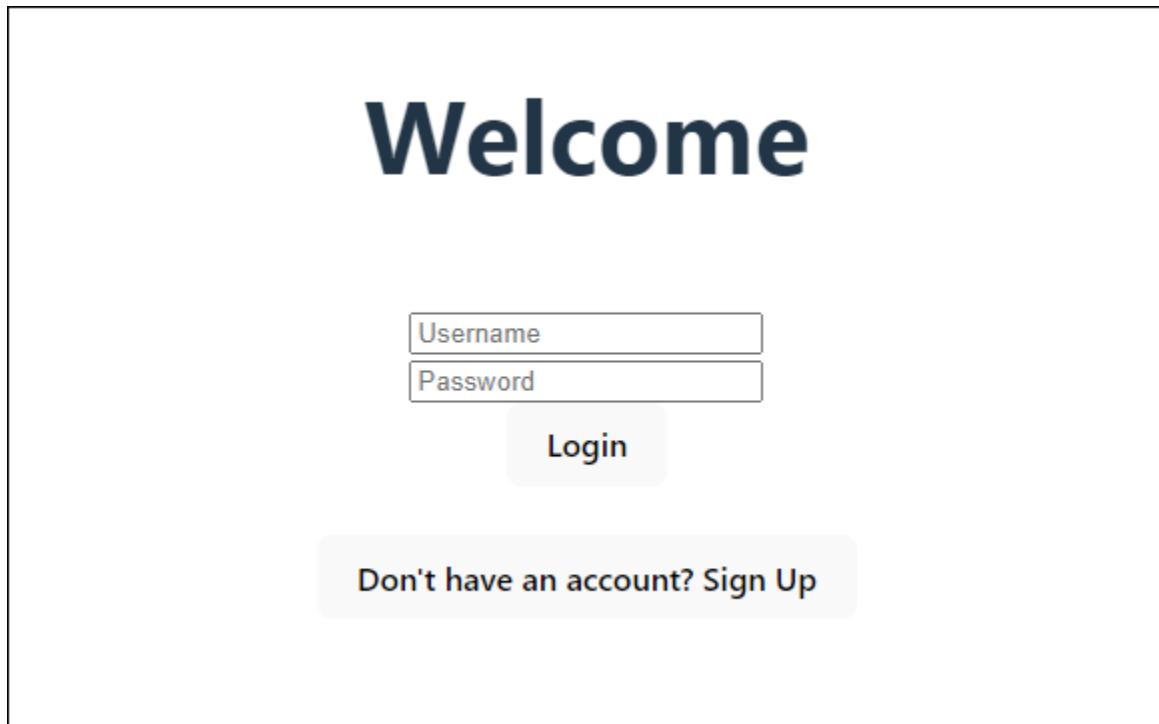
Topics

- [Set up an example React single page application](#)
- [Set up an example Android app with Flutter](#)

Set up an example React single page application

In this tutorial, you'll create a React single page application where you can test user sign-up, confirmation, and sign-in. React is a JavaScript-based library for web and mobile apps, with a focus on the user interface (UI). This example application demonstrates some basic functions of Amazon Cognito user pools. If you're already experienced in web app development with React, [download the example app from GitHub](#).

The following screenshot is of the initial authentication page in the application that you'll create.



To set up this application, your user pool must meet the following requirements:

- Users can sign in with their email address. **Cognito user pool sign-in options: Email**.
- Usernames are case insensitive. **User name requirements: Make user name case sensitive** is not selected.
- Multi-factor authentication (MFA) isn't required. **MFA enforcement: Optional MFA**.
- Your user pool verifies attributes for user-profile confirmation with an email message. **Attributes to verify: Send email message, verify email address**.

- Email is the only required attribute. **Required attributes:** email.
- Users can sign themselves up in your user pool. **Self-registration: Enable self-registration** is selected.
- Your initial app client is a public client that permits sign-in with username and password. **App type: Public client, Authentication flows:** ALLOW_USER_PASSWORD_AUTH.

Create an application

To build this application, you must set up a developer environment. The developer environment requirements are:

1. Node.js is installed and updated.
2. Node package manager (npm) is installed and updated to at least version 10.2.3.
3. The environment is accessible on TCP port 5173 in a web browser.

To create an example React web application

1. Sign in to your developer environment and navigate to the parent directory for your application.

```
cd ~/path/to/project/folder/
```

2. Create a new React service.

```
npm create vite@latest frontend-client -- --template react-ts
```

3. Clone the cognito-developer-guide-react-example [project folder](#) from the AWS code examples repository on GitHub.

```
cd ~/some/other/path
```

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

```
cp -r ./aws-doc-sdk-examples/javascriptv3/example_code/cognito-identity-provider/scenarios/cognito-developer-guide-react-example/frontend-client ~/path/to/project/folder/frontend-client
```

4. Navigate to the `src` directory in your project.

```
cd ~/path/to/project/folder/frontend-client/src
```

5. Edit `config.json` and replace the following values:

- Replace `YOUR_AWS_REGION` with an AWS Region code. For example: `us-east-1`.
 - Replace `YOUR_COGNITO_USER_POOL_ID` with the ID of the user pool that you have designated for testing. For example: `us-east-1_EXAMPLE`. The user pool must be in the AWS Region that you entered in the previous step.
 - Replace `YOUR_COGNITO_APP_CLIENT_ID` with the ID of the app client that you have designated for testing. For example: `1example23456789`. The app client must be in the user pool from the previous step.
6. If you want to access your example application from an IP other than `localhost`, edit `package.json` and change the line `"dev": "vite"`, to `"dev": "vite --host 0.0.0.0",`.
 7. Install your application.

```
npm install
```

8. Launch the application.

```
npm run dev
```

9. Access the application in a web browser at `http://localhost:5173` or `http://[IP address]:5173`.
10. Sign up a new user with a valid email address.
11. Retrieve the confirmation code from your email message. Enter the confirmation code into the application.
12. Sign in with your username and password.

Creating a React developer environment with Amazon Lightsail

A quick way to get started with this application is to create a virtual cloud server with Amazon Lightsail.

With Lightsail, you can quickly create a small server instance that comes preconfigured with the prerequisites for this example application. You can SSH to your instance with a browser-based client, and connect to the web server at a public or private IP address.

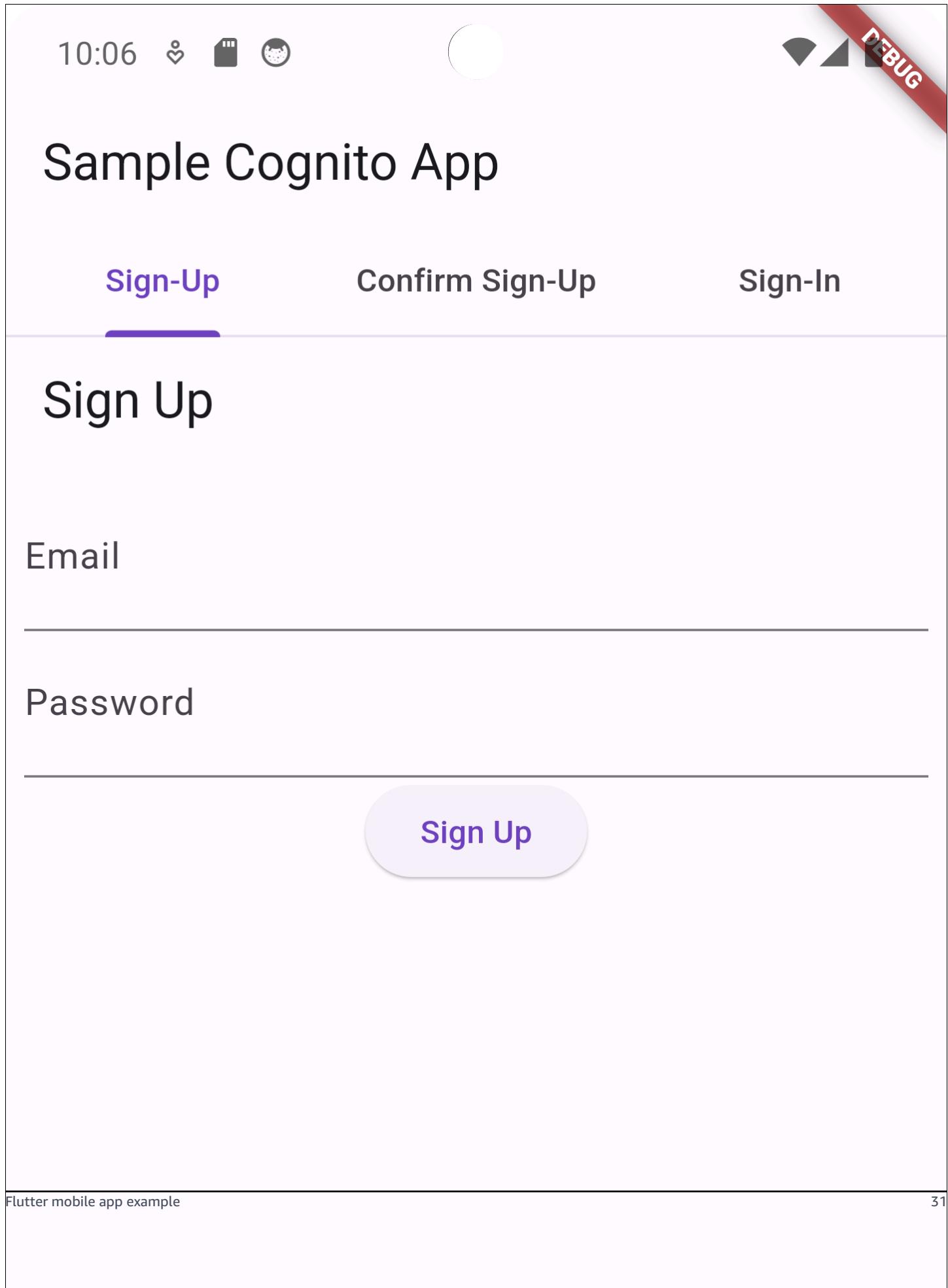
To create a Lightsail instance for this example application

1. Go to the [Lightsail console](#). If prompted, enter your AWS credentials.
2. Choose **Create instance**.
3. For **Select a platform**, choose **Linux/Unix**.
4. For **Select a blueprint**, choose **Node.js**.
5. Under **Identify your instance**, give your development environment a friendly name.
6. Choose **Create instance**.
7. After Lightsail has created your instance, select it and from the **Connect** tab, choose **Connect using SSH**.
8. An SSH session opens in a browser window. Run `node -v` and `npm -v` to confirm that your instance was provisioned with Node.js and the minimum npm version of 10.2.3.
9. Proceed to [configure your React application](#).

Set up an example Android app with Flutter

In this tutorial, you'll create a mobile application in Android Studio where you can emulate a device and test user sign-up, confirmation, and sign-in. This example application creates a basic Amazon Cognito user pools mobile client for Android in Flutter. If you're already experienced in mobile app development with Flutter, [download the example app from GitHub](#).

The following screenshot shows the app running on a virtual Android device.



To set up this application, your user pool must meet the following requirements:

- Users can sign in with their email address. **Cognito user pool sign-in options: Email.**
- Usernames are case insensitive. **User name requirements: Make user name case sensitive** is not selected.
- Multi-factor authentication (MFA) isn't required. **MFA enforcement: Optional MFA.**
- Your user pool verifies attributes for user-profile confirmation with an email message. **Attributes to verify: Send email message, verify email address.**
- Email is the only required attribute. **Required attributes: email.**
- Users can sign themselves up in your user pool. **Self-registration: Enable self-registration** is selected.
- Your initial app client is a public client that permits sign-in with username and password. **App type: Public client, Authentication flows: ALLOW_USER_PASSWORD_AUTH.**

Create an application

To create an example Android app

1. Install [Android studio](#) and [command-line tools](#).
2. In Android Studio, install the [Flutter plugin](#).
3. Create a new Android Studio project from the contents of the `cognito_flutter_mobile_app` directory in [this example app](#).
 - Edit `assets/config.json` and replace `<<YOUR USER POOL ID>>` and `<< YOUR CLIENT ID>>` with the IDs of your user pool and app client.
4. Install [Flutter](#).
 - a. Add Flutter to your PATH variable.
 - b. Accept licenses with the following command.

`flutter doctor --android-licenses`
 - c. Verify your Flutter environment and install any missing components.

`flutter doctor`

- If any components are missing, run `flutter doctor -v` to learn how to fix the issue.
- d. Change to the directory of your new Flutter project and install dependencies.
- Run `flutter pub add amazon_cognito_identity_dart_2`.
 - e. Run `flutter pub add flutter_secure_storage`.
5. Create a virtual Android device.
1. In the Android studio GUI, create a new device with the [device manager](#).
 2. In the CLI, run `flutter emulators --create --name android-device`.
6. Launch your virtual Android device.
1. In the Android Studio GUI, select the start  icon next to your virtual device.
 2. In the CLI, run `flutter emulators --launch android-device`.
7. Launch your app on your virtual device.
1. In the Android Studio GUI, select the deploy  icon.
 2. In the CLI, run `flutter run`.
8. Navigate to your running virtual device in Android Studio.
9. Sign up a new user with a valid email address.
10. Retrieve the confirmation code from your email message. Enter the confirmation code into the application.
11. Sign in with your username and password.

Add more features and security options to your user pool

After you have followed the tutorials to complete example applications, you can broaden the scope of your user pool implementation. Or, if you didn't create a test application, create a new user pool according to your preferences. You can customize user pool features for other applications or [add](#)

[external identity providers](#). As you plan your move to put Amazon Cognito user pools in production applications, you can evaluate [additional examples and tutorials](#).

Amazon Cognito has feature plans that add functional and security options when you opt in to higher tiers. You can start with the *Lite* plan, add advanced authentication and authorization options with the *Essentials* plan, and add automated-reasoning security guardrails with the *Plus* plan. For more information, see [User pool feature plans](#).

The following are some additional Amazon Cognito user pools features:

- [Apply branding to managed login pages](#)
- [Adding MFA to a user pool](#)
- [Advanced security with threat protection](#)
- [Customizing user pool workflows with Lambda triggers](#)
- [Using Amazon Pinpoint for user pool analytics](#)

For an overview of Amazon Cognito authentication and authorization models, see [How authentication works with Amazon Cognito](#).

To access other AWS services after a successful user pool authentication, see [Accessing AWS services using an identity pool after sign-in](#).

In addition to using the AWS Management Console and the user pool SDKs, you can also manage your user pools by using the [AWS Command Line Interface](#).

Topics

- [Add social sign-in to your user pool](#)
- [Add a SAML 2.0 identity provider](#)

Add social sign-in to your user pool

Providing users with the ability to sign in to your application through their existing public, or social, identity providers can improve their authentication experience. Amazon Cognito user pools integrate with popular social identity providers (IdPs) like Facebook, Google, Amazon, and Apple, giving your users convenient sign-in options that they are already familiar with.

When you set up social sign-in, you are giving your users an alternative to creating a dedicated account just for your application. This can improve conversion rates and make the sign-up process

more seamless. From the user's perspective, they can apply their existing social credentials to quickly authenticate, without the friction of remembering another username and password.

Configuring a social IdP in your user pool involves a few key steps. You must register your application with the social provider to obtain a client ID and secret. Then you can add the social IdP configuration to your user pool, specifying the scopes that you want to request and the user pool attributes that you want to map from IdP attributes. At runtime, Amazon Cognito handles the token exchange with the provider, maps user attributes, and issues tokens to your application in the shared user pool format.

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**.

If you don't have an existing Facebook app, you will see a different option. Choose **Create App**.

4. On the **Create an app** page, choose a use case for your app, and then choose **Next**.
5. Enter a name for your Facebook app and choose **Create App**.
6. On the left navigation bar, choose **App Settings**, and then choose **Basic**.
7. Note the **App ID** and the **App Secret**. You will use them in the next section.
8. Choose **+ Add platform** from the bottom of the page.
9. On the **Select Platform** screen, select your platforms, and then choose **Next**.
10. Choose **Save changes**.
11. For **App Domains**, enter your user pool domain.

`https://your_user_pool_domain`

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

Enter your redirect URL into **Valid OAuth Redirect URIs**. The redirect URL consists 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 the navigation bar at the top of the page, and then choose **Login with Amazon**.

4. Choose **Create a Security Profile**.
5. Enter 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 icon and choose **Web Settings**, and then choose **Edit**.
9. Enter your user pool domain into **Allowed Origins**.

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

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

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

11. Choose **Save**.

To register an app with Google

For more information about OAuth 2.0 in the Google Cloud platform, see [Learn about authentication & authorization](#) in the Google Workspace for Developers documentation.

1. Create a [developer account with Google](#).
2. Sign in to the [Google Cloud Platform console](#).
3. From the top navigation bar, choose **Select a project**. If you already have a project in the Google platform, this menu displays your default project instead.
4. Select **NEW PROJECT**.
5. Enter a name for your product and then choose **CREATE**.
6. On the left navigation bar, choose **APIs and Services**, and then choose **Oauth consent screen**.
7. Enter the app information, an **App domain**, **Authorized domains**, and **Developer contact information**. Your **Authorized domains** must include amazoncognito.com and the root of your custom domain. For example: example.com. Choose **SAVE AND CONTINUE**.
8. 1. Under **Scopes**, choose **Add or remove scopes**, and then choose, at a minimum, the following OAuth scopes.
 1./auth/userinfo.email
 2./auth/userinfo.profile
 3. openid
9. Under **Test users**, choose **Add users**. Enter your email address and any other authorized test users, and then choose **SAVE AND CONTINUE**.
10. Expand the left navigation bar again, choose **APIs and Services**, and then choose **Credentials**.
11. Choose **CREATE CREDENTIALS**, and then choose **OAuth client ID**.
12. Choose an **Application type** and give your client a **Name**.
13. Under **Authorized JavaScript origins**, choose **ADD URI**. Enter your user pool domain.

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

14. Under **Authorized redirect URIs**, choose **ADD URI**. Enter the path to the /oauth2/idpresponse endpoint of your user pool domain.

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

15. Choose **CREATE**.

16. Securely store the values that Google displays under **Your client ID** and **Your client secret**. Provide these values to Amazon Cognito when you add a Google IdP.

To register an app with Apple

For more information about setting up Sign in with Apple, see [Configuring Your Environment for Sign in with Apple](#) in the Apple Developer documentation.

1. Create a [developer account with Apple](#).
2. [Sign in](#) with your Apple credentials.
3. On the left navigation bar, choose **Certificates, Identifiers & 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 **Select a type** page, choose **App**, and then choose **Continue**.
8. On the **Register an App ID** page, do the following:
 1. Under **Description**, enter a description.
 2. Under **App ID Prefix**, enter a **Bundle ID**. Make a note of the value under **App ID Prefix**. You will use this value after you choose Apple as your identity provider in [Step 2: Add a social IdP to your user pool](#).
 3. Under **Capabilities**, choose **Sign In with Apple**, and then choose **Edit**.
 4. On the **Sign in with Apple: App ID Configuration** page, choose to set up the app as either primary or grouped with other App IDs, and then choose **Save**.
 5. Choose **Continue**.
9. On the **Confirm your App ID** page, choose **Register**.
10. On the **Identifiers** page, choose the + 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:
 1. Under **Description**, enter a description.
 2. Under **Identifier**, enter an identifier. Make a note of this Services ID because you'll need this value after you choose Apple as your identity provider in [Step 2: Add a social IdP to your user pool](#).

3. Choose **Continue** and then choose **Register**.
 13. Choose the Services ID that you just created from the Identifiers page.
 1. Select **Sign In with Apple**, and then choose **Configure**.
 2. On the **Web Authentication Configuration** page, select the app ID that you created earlier as the **Primary App ID**.
 3. Choose the + icon next to **Website URLs**.
 4. Under **Domains and subdomains**, enter your user pool domain without an https:// prefix.

<your-user-pool-domain>

 - 5. Under **Return URLs**, enter the path to the /oauth2/idpresponse endpoint of your user pool domain.
- https://<your-user-pool-domain>/oauth2/idpresponse*
6. Choose **Next**, and then choose **Done**. You don't need to verify the domain.
 7. Choose **Continue**, and then choose **Save**.
 14. On the left navigation bar, choose **Keys**.
 15. On the **Keys** page, choose the + icon.
 16. On the **Register a New Key** page, do the following:
 1. Under **Key Name**, enter a key name.
 2. Choose **Sign In with Apple**, and then choose **Configure**.
 3. On the **Configure Key** page, select the app ID that you created earlier as the **Primary App ID**. Choose **Save**.
 4. Choose **Continue**, and then choose **Register**.
 17. On the **Download Your Key** page, choose **Download** to download the private key, note the **Key ID** shown, 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](#).

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 **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Social and external providers** menu. Locate **Federated sign-in** and select **Add an identity provider**.
5. Choose a social identity provider: **Facebook**, **Google**, **Login with Amazon**, or **Sign in with Apple**.
6. Choose from the following steps, based on your choice of social identity provider:
 - **Google and Login with Amazon** – Enter the **app client ID** and **app client secret** that was generated in the previous section.
 - **Facebook** – Enter the **app client ID** and **app client secret** that was generated in the previous section, and then choose an API version (for example, version 2.12). We recommend choosing the latest possible version—each Facebook API has a lifecycle and deprecation date. Facebook scopes and attributes can vary between API versions. We recommend testing your social identity log in with Facebook to ensure that federation works as intended.
 - **Sign in with Apple** – Enter the **Services ID**, **Team ID**, **Key ID**, and **private key** that was generated in the previous section.
7. Enter the names of the **Authorized scopes** that you want to use. 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

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

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

With 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 (these can be from internal failures within Amazon 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 user signs in. Users only retrieve new information when they authenticate and when they refresh their tokens.
 - A developer deletes the user and then the user signs in again without removing the app from their Apple ID profile.
8. Map attributes from your identity provider to your user pool. For more information, see [Things to know about mappings](#).
 9. Choose **Create**.
 10. From the **App clients** menu, choose one of the app clients in the list and **Edit hosted UI settings**. Add the new social identity provider to the app client under **Identity providers**.
 11. Choose **Save changes**.

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://mydomain.us-east-1.amazonaws.com/login?  
response_type=code&client_id=1example23456789&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**

Amazon Cognito cancels authentication requests that do not complete within 5 minutes, and redirects the user to managed login. The page displays a Something went wrong error message.

Add a SAML 2.0 identity provider

Your app users can sign in with a SAML 2.0 identity provider (IdP). You might choose SAML 2.0 IdPs over social IdPs when your customers are the internal customers or linked businesses of your organization. Where a social IdP permits all users to register for an account, a SAML IdP is more likely to pair with a user directory that your organization controls. 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.

For more information, see [Using SAML identity providers with a user pool](#).

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

You must also provide an assertion consumer service (ACS) endpoint to your SAML identity provider. Configure the following endpoint in your user pool domain for SAML 2.0 POST binding in your SAML identity provider. For more information about user pool domains, see [Configuring a user pool domain](#).

`https://Your user pool domain/saml2/idpresponse`

With an Amazon Cognito domain:

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

With a custom domain:

`https://Your custom domain/saml2/idpresponse`

You can find your domain prefix and the Region value for your user pool in the **Domain** menu in the [Amazon Cognito console](#).

For some SAML identity providers, you also need to provide the service provider (SP) `urn`, also called the audience URI or SP entity ID, in the format:

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

You can find your user pool ID in the **Overview** dashboard for your user pool 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 through a user agent.

To configure a SAML 2.0 identity provider in your user pool

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Social and external providers** menu. Locate **Federated sign-in** and select **Add an identity provider**.
5. Choose a **SAML** social identity provider.
6. Enter **Identifiers** separated by commas. An identifier tells Amazon Cognito it should check the email address that a user enters when they sign in. Then it directs them to the provider that corresponds to their domain.
7. Choose **Add sign-out flow** if you want Amazon Cognito to send signed sign-out requests to your provider when a user logs out. You must configure your SAML 2.0 identity provider to send sign-out responses to the `https://<your Amazon Cognito domain>/saml2/logout` endpoint that is created when you configure managed login. The `saml2/logout` endpoint uses the POST binding.

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 that is provided by Amazon Cognito with your SAML IdP.

The SAML IdP will process the signed logout request and will log out your user from the Amazon Cognito session.

8. Choose a **Metadata document source**. If your identity provider offers SAML metadata at a public URL, you can choose **Metadata document URL** and enter that public URL. Otherwise, choose **Upload metadata document** and select a metadata file you downloaded from your provider earlier.

Note

We recommend that you enter a metadata document URL if your provider has a public endpoint, rather than uploading a file. This allows Amazon Cognito to refresh the metadata automatically. Typically, metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

9. Select **Map attributes between your SAML provider and your app** to map SAML provider attributes to the user profile in your user pool. Include your user pool required attributes in your attribute map.

For example, when you choose the **User pool attribute** `email`, enter the SAML attribute name as it appears in the SAML assertion from your identity provider. Your identity provider might offer sample SAML assertions for reference. Some identity providers use simple names, such as `email`, while others use URL-formatted attribute names, such as the following example:

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

10. Choose **Create**.

Getting started with Amazon Cognito identity pools

With Amazon Cognito identity pools, you can create unique identities and assign permissions for users. Your identity pool can bring in identities from the following types of authentication services:

- Users in an Amazon Cognito user pool
- Users who authenticate with external identity providers such as Facebook, Google, Apple, or an OIDC or SAML identity provider.
- Users authenticated via your own existing authentication process

After users authenticate with their provider and present authorization to an identity pool, they get temporary AWS credentials. Users' credentials have permissions that you define for access to other AWS services.

Topics

- [Create an identity pool in Amazon Cognito](#)
- [Set up an SDK](#)
- [Integrate the identity providers](#)
- [Get credentials](#)

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. The following procedure is a general guide to create a new identity pool in the console. You can also [skip straight to the console](#) and follow the guided experience and inline help content.

To create a new identity pool in the console

1. Sign in to the [Amazon Cognito console](#) and select **Identity pools**.
2. Choose **Create identity pool**.
3. In **Configure identity pool trust**, choose to set up your identity pool for **Authenticated access**, **Guest access**, or both.

- If you chose **Authenticated access**, select one or more **Identity types** that you want to set as the source of authenticated identities in your identity pool. If you configure a **Custom developer provider**, you can't modify or delete it after you create your identity pool.
4. In **Configure permissions**, choose a default IAM role for authenticated or guest users in your identity pool.
- a. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
 - b. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see [Role trust and permissions](#).
5. In **Connect identity providers**, enter the details of the identity providers (IdPs) that you chose in **Configure identity pool trust**. You might be asked to provide OAuth app client information, choose an Amazon Cognito user pool, choose an IAM IdP, or enter a custom identifier for a developer provider.
- a. Choose the **Role settings** for each IdP. You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With an Amazon Cognito user pool IdP, you can also **Choose role with preferred_role in tokens**. For more information about the `cognito:preferred_role` claim, see [Assigning precedence values to groups](#).
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

- b. Configure **Attributes for access control** for each IdP. Attributes for access control maps user claims to [principal tags](#) that Amazon Cognito applies to their temporary session. You can build IAM policies to filter user access based on the tags that you apply to their session.
 - i. To apply no principal tags, choose **Inactive**.
 - ii. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - iii. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
6. In **Configure properties**, enter a **Name** under **Identity pool name**.
 7. Under **Basic (classic) authentication**, choose whether you want to **Activate basic flow**. With the basic flow active, you can bypass the role selections you made for your IdPs and call [AssumeRoleWithWebIdentity](#) directly. For more information, see [Identity pools authentication flow](#).
 8. Under **Tags**, choose **Add tag** if you want to apply [tags](#) to your identity pool.
 9. In **Review and create**, confirm the selections that you made for your new identity pool. Select **Edit** to return to the wizard and change any settings. When you're done, select **Create identity pool**.

Set up an SDK

To use Amazon Cognito identity pools, set up AWS Amplify, the AWS SDK for Java, or the AWS SDK for .NET. For more information, see the following topics.

- [Setting up the SDK for JavaScript](#) in the *AWS SDK for JavaScript Developer Guide*
- [Amplify Documentation](#) in the *Amplify Dev Center*
- [Amazon Cognito credentials provider](#) in the *AWS SDK for .NET Developer Guide*

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—and unauthenticated identities. This feature also supports [Developer-](#)

[authenticated identities](#), which lets you register and authenticate users via your own backend authentication process.

To learn more about using an Amazon Cognito user pool to create your own user directory, see [Amazon Cognito user pools](#) and [Accessing AWS services using an identity pool after sign-in](#).

To learn more about using external identity providers, see [Identity pools third-party identity providers](#).

To learn more about integrating your own backend authentication process, see [Developer-authenticated identities](#).

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 backend in AWS or outside AWS through Amazon API Gateway. See [Getting credentials](#).

Guided setup options for Amazon Cognito

You might want to evaluate the features of Amazon Cognito in a structured, guided experience. Here are some external resources that provide tailored experiences with user pools and identity pools.

Complete a workshop

AWS workshop studio [hosts a workshop](#) that walks you through the setup of the majority of Amazon Cognito features. These features include the user pools API, the user pools hosted UI, identity pools, and security configuration.

Add application code from examples

The [code examples](#) chapter in this guide has application code that you can use with user pools and identity pools. The user pools section of the code examples chapter has short snippets that cover individual operations, and longer examples for end-to-end example applications in a variety of programming languages.

Create a fullstack application with AWS Amplify

[AWS Amplify](#) is an AWS service for developers who want to develop and host an application and user interface. Amazon Cognito is the authentication component of Amplify. When you add authentication to your application, Amplify can automate the deployment of Amazon Cognito user pool and identity pool resources. Also, see [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#).

More Amazon Cognito application resources on GitHub

- [Authentication flow examples with .NET for Amazon Cognito](#)
- [Amazon Cognito Passwordless Auth](#)
- [PetStore example with Amazon Verified Permissions](#)
- [Sample React App Using ABAC + Identity Pools to Access AWS Resources](#)
- [Amazon Cognito and API Gateway based machine to machine authorization using AWS CDK](#)
- [Building fine-grained authorization using Amazon Cognito, API Gateway, and IAM](#)
- [CloudFront authorization@edge](#)

More workshops

- [Implement Passwordless authentication with Amazon Cognito and WebAuthn](#)
- [Multi-tenant SaaS identity with Amazon Cognito user pools](#)
- [Amazon Cognito JWT Deep Dive](#)

Blogposts

- [Protect public clients for Amazon Cognito by using an Amazon CloudFront proxy](#)
- [How to set up Amazon Cognito for federated authentication using Azure AD](#)
- [Simplify web app authentication: A guide to AD FS federation with Amazon Cognito user pools](#)

Integrating Amazon Cognito authentication and authorization with web and mobile apps

The lowest-effort integration you can create with Amazon Cognito user pools is with [managed login](#). User pool authentication with managed login requires OpenID Connect (OIDC) libraries that direct users to hosted sign-in pages. In this series of user-interactive and redirect web endpoints, Amazon Cognito handles the flow of authentication, including third-party sign-in, multi-factor authentication (MFA), and choosing an authentication flow. Your application only has to process the authentication outcome that Amazon Cognito returns in the response.

You can also add an AWS SDK to your application, custom-build authentication interfaces, and invoke API operations for authentication and authorization of your users. [AWS Amplify](#) is an AWS service for building full-stack applications, with Amazon Cognito authentication in the back end.

Topics

- [Authentication with AWS Amplify](#)
- [Authentication with AWS SDKs](#)
- [How authentication works with Amazon Cognito](#)
- [Using this service with an AWS SDK](#)
- [Authorization with Amazon Verified Permissions](#)

Implementation of Amazon Cognito is a mix of AWS Management Console or AWS SDK administrative tools, and SDK libraries in applications. The Amazon Cognito console is the visual interface for setup and management of your Amazon Cognito user pools and identity pools.

Managed login is a ready-to-use web-based sign-in application for quick testing and deployment of Amazon Cognito user pools. It requires configuration of OIDC libraries in your apps to interact with your user pools. For example, your app might invoke managed login for user sign-in, then call the token endpoint from your app code to exchange your user's authorization code for tokens. Then your app must interpret and store your user's tokens, and present them in the appropriate context for authentication and authorization. Amplify adds guided integration tools with built-in functions for these processes.

You can also build your Amazon Cognito resources entirely in code. Identity pools don't have the same managed authentication options as user pools—for access to AWS credentials in your

applications, implement identity pools operations in imported SDK modules. To get started with your own custom-built application code, visit the Amazon Cognito [code examples for AWS SDKs](#). For integration with the Amazon Cognito as an OpenID Connect identity provider, use [OpenID Connect developer tools](#).

Before you use Amazon Cognito authentication and authorization, choose an app platform and prepare your code to integrate with the service. For available platforms for AWS SDKs, see [Authentication with AWS SDKs](#). The AWS CLI is a command-line SDK for Amazon Cognito and other AWS services, and is a valuable place to begin to familiarize yourself with Amazon Cognito API operations and their syntax.

 **Note**

Some components of Amazon Cognito can be configured only with the API. For example, you can only set a user pool [custom SMS or email sender](#) Lambda trigger with a request that updates the LambdaConfig property of the [UserPool](#) class in a [CreateUserPool](#) or [UpdateUserPool](#) API request.

The Amazon Cognito user pools API shares its namespace with several classes of API operations. One class configures user pools and their processes, identity providers and users. Another includes unauthenticated operations for your users in a public client to sign in, sign out, and manage their profiles. The final class of API operations performs user operations that you authorize with your own AWS credentials in a confidential server-side client. You must know your intended app architecture before you begin to implement app code. For more information, see [Understanding API, OIDC, and managed login pages authentication](#).

Authentication with AWS Amplify

AWS Amplify is a complete solution for building web and mobile applications. With Amplify, you can connect to existing resources with the Amplify libraries, or you can create and configure new resources with the Amplify command line interface (CLI). Amplify also has connected UI components like [Authenticator](#) for setup and customization of the sign-in and sign-up experience in your app.

To use Amplify authentication features in your front-end app, see the following documentation by platform.

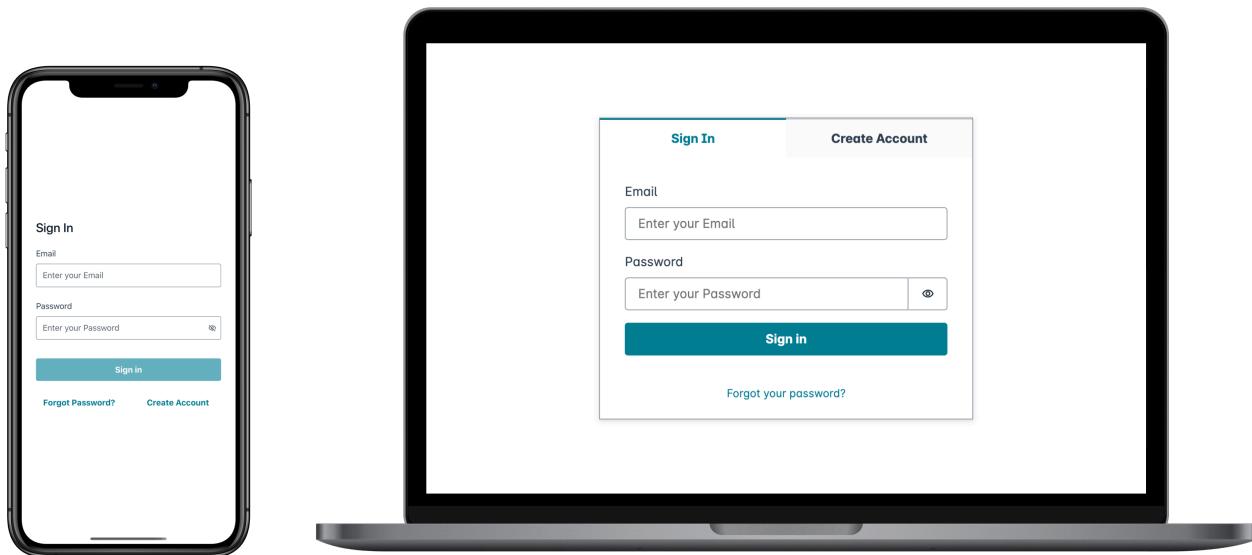
- [Amplify authentication for JavaScript](#)
- [Amplify authentication for iOS](#)
- [Amplify authentication for Android](#)
- [Amplify authentication for Flutter](#)

The Amplify libraries are open source and are available on [GitHub](#). To learn more about how Amplify Auth implements Amazon Cognito authentication, visit the following libraries.

- [amplify-js](#)
- [amplify-swift](#)
- [amplify-flutter](#)
- [amplify-android](#)

Creating a user interface (UI) with Amplify

[User pool managed login](#) can fulfill the essential needs of an authentication front-end for a web or mobile app. To customize your user interface (UI) beyond the parameters that managed login accommodates, custom-build an application. [Amplify UI](#) is a customizable collection of front-end components in a variety of languages.



To get started with your custom authentication component, visit the following documentation for the Authenticator component.

- [Authenticator for Android](#)
- [Authenticator for Angular](#)
- [Authenticator for Flutter](#)
- [Authenticator for React](#)
- [Authenticator for React Native](#)
- [Authenticator for Swift](#)
- [Authenticator for Vue](#)

Authentication with AWS SDKs

To use a secure backend to build your own identity microservice that interacts with Amazon Cognito, connect to the Amazon Cognito user pools and Amazon Cognito identity pools API with an AWS SDK in the language of your choice.

For details on each API operation, see the [Amazon Cognito user pools API Reference](#) and the [Amazon Cognito API Reference](#). These documents contain [See also](#) sections with resources for using a variety of SDKs in supported platforms.

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

How authentication works with Amazon Cognito

When your customer signs in to an Amazon Cognito user pool, your application receives JSON web tokens (JWTs).

When your customer signs in to an identity pool, either with a user pool token or another provider, your application receives temporary AWS credentials.

With user pool sign-in, you can implement authentication and authorization entirely with an AWS SDK. If you don't want to build your own user interface (UI) components, you can invoke a prebuilt web UI (managed login) or the sign-in page for your third-party identity provider (IdP).

This topic is an overview of some of the ways that your application can interact with Amazon Cognito to authenticate with ID tokens, authorize with access tokens, and access AWS services with identity pool credentials.

Topics

- [User pool authentication with managed login](#)
- [User pool API authentication and authorization with an AWS SDK](#)
- [User pool authentication with a third-party identity provider](#)
- [Identity pool authentication](#)

User pool authentication with managed login

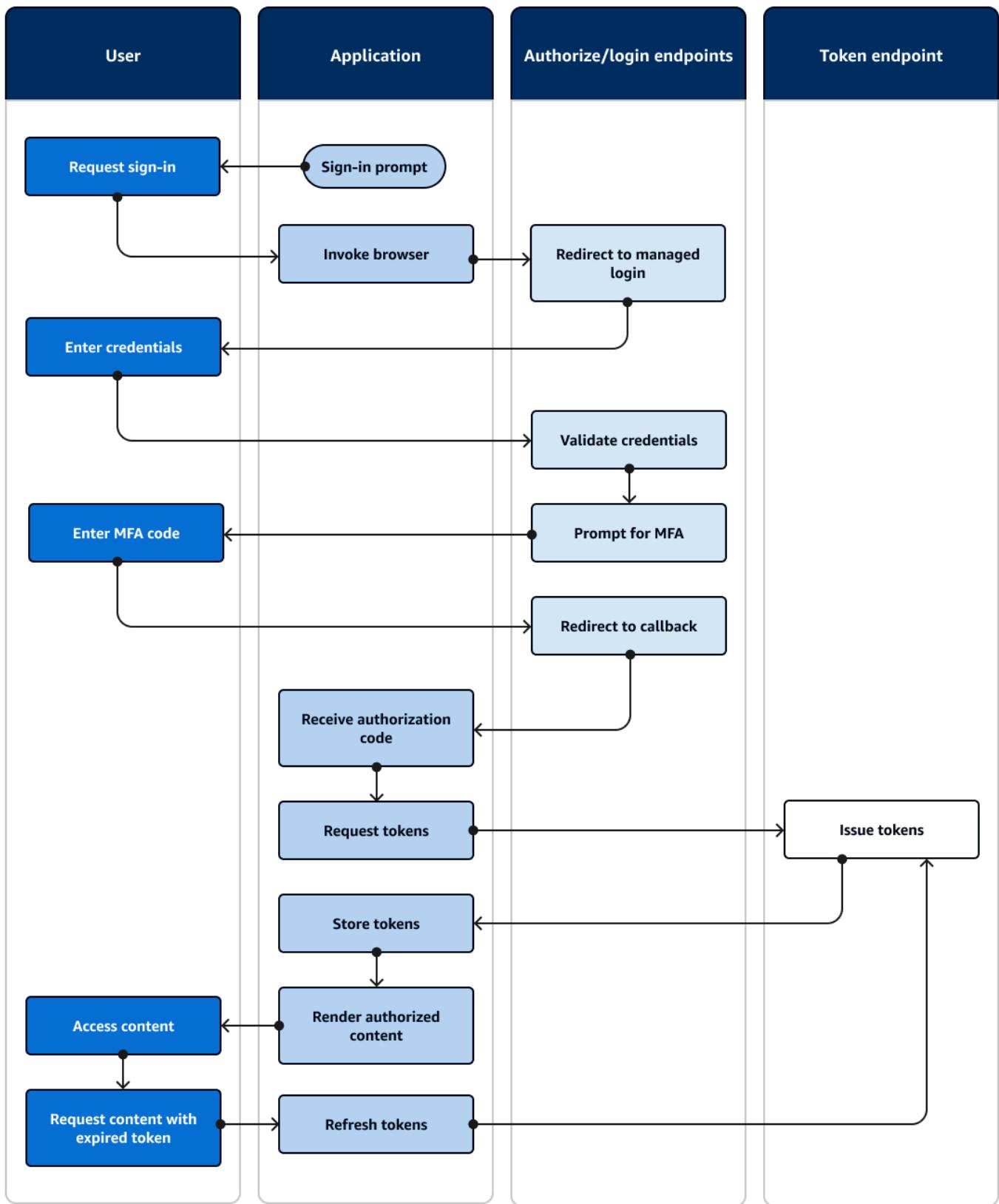
[Managed login](#) is a website that is linked to your user pool and app client. It can perform sign-in, sign-up, and password-reset operations for your users. An application with a managed login component for authentication can require less developer effort to implement. An application can skip UI components for authentication and invoke managed login webpages in the user's browser.

Applications collect users' JWTs with a web or app redirect location. Applications that implement managed login can connect to user pools for authentication as if they were an OpenID Connect (OIDC) IdP.

Managed login authentication fits the model where applications need an authorization server, but don't need features like custom authentication, identity pools integration, or user attribute self-service. When you want to use some of these advanced options, you can implement them with a user pools component for an SDK.

Managed login and third-party IdP authentication models, with a primary reliance on OIDC implementation, are best for advanced authorization models with OAuth 2.0 scopes.

The following diagram illustrates a typical sign-in session for managed login authentication.



Managed login authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. The application directs the user to a sign-in prompt in the managed login pages of your user pool domain.
4. They enter their username and password.
5. The user pool validates the user's credentials and determines that the user has activated multi-factor authentication (MFA).
6. The managed login page prompts the user to enter an MFA code.
7. The user enters their MFA code.
8. Your user pool redirects the user to the application URL.
9. The application collects the authorization code from the URL request parameter that managed login appended to the [callback URL](#).
10. The application requests tokens with the authorization code.
11. The token endpoint returns JWTs to the application.
12. The application decodes, validates, and stores or caches the user's JWTs.
13. The application displays the requested access-controlled component.
14. The user views their content.
15. Later, the user's access token has expired, and they request to view an access-controlled component.
16. The application determines that the user's session should persist. It requests new tokens from the token endpoint with the refresh token.

Variants and customization

You can customize the look and feel of your managed login pages with the [branding designer](#) for your entire user pool, or at the level of any [app client](#). You can also [configure app clients](#) with their own identity providers, scopes, access to user attributes, and advanced security configuration.

Related resources

- [User pool managed login](#)
- [Scopes, M2M, and APIs with resource servers](#)

- [User pool endpoints and managed login reference](#)

User pool API authentication and authorization with an AWS SDK

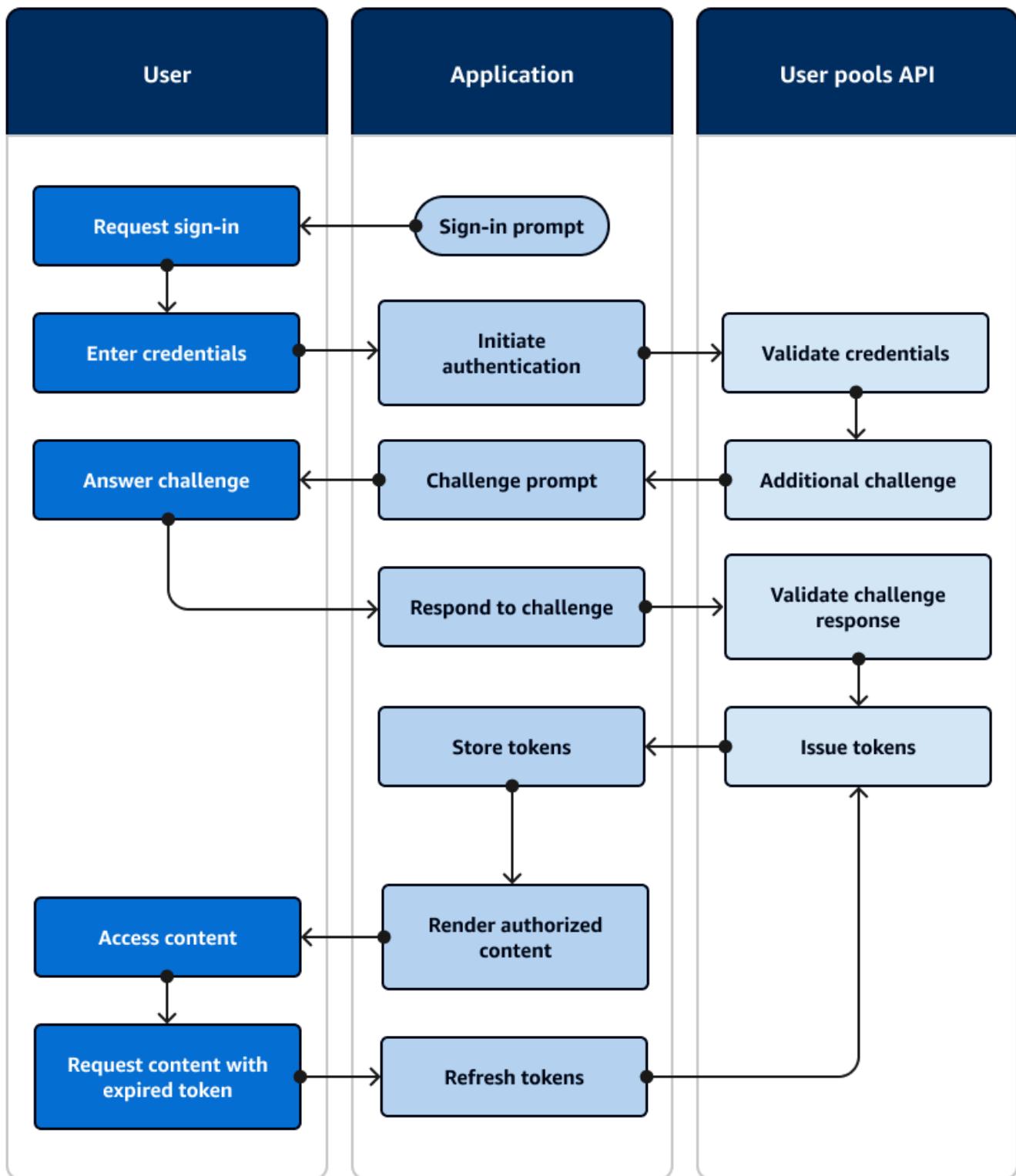
AWS has developed components for Amazon Cognito user pools, or *Amazon Cognito identity provider*, in [a variety of developer frameworks](#). The methods built into these SDKs call the [Amazon Cognito user pools API](#). The same user pools API namespace has operations for configuration of user pools and for user authentication. For a more thorough overview, see [Understanding API, OIDC, and managed login pages authentication](#).

API authentication fits the model where your applications have existing UI components and primarily rely on the user pool as a user directory. This design adds Amazon Cognito as a component within a larger application. It requires programmatic logic to handle complex chains of challenge and response.

This application doesn't need to implement a full OpenID Connect (OIDC) relying party implementation. Instead, it has the ability to decode and use JWTs. When you want access to the full set of user pool features for [local users](#), build your authentication with the Amazon Cognito SDK in your development environment.

API authentication with custom OAuth scopes is less oriented toward external API authorization. To add custom scopes to an access token from API authentication, modify the token at runtime with a [Pre token generation Lambda trigger](#).

The following diagram illustrates a typical sign-in session for API authentication.



API authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. They enter their username and password.
4. The application invokes the method that makes an [InitiateAuth](#) API request. The request passes the user's credentials to a user pool.
5. The user pool validates the user's credentials and determines that the user has activated multi-factor authentication (MFA).
6. The user pool responds with a challenge that requests an MFA code.
7. The application generates a prompt that collects the MFA code from the user.
8. The application invokes the method that makes a [RespondToAuthChallenge](#) API request. The request passes the user's MFA code.
9. The user pool validates the user's MFA code.
10. The user pool responds with the user's JWTs.
11. The application decodes, validates, and stores or caches the user's JWTs.
12. The application displays the requested access-controlled component.
13. The user views their content.
14. Later, the user's access token has expired, and they request to view an access-controlled component.
15. The application determines that the user's session should persist. It invokes the [InitiateAuth](#) method again with the refresh token and retrieves new tokens.

Variants and customization

You can augment this flow with additional challenges—for example, your own custom authentication challenges. You can automatically restrict access for users whose passwords have been compromised, or whose unexpected sign-in characteristics might indicate a malicious sign-in attempt. This flow looks much the same for operations to sign up, update user attributes, and reset passwords. Most of these flows have duplicate public (client-side) and confidential (server-side) API operations.

Related resources

- [Amazon Cognito user pools API](#)

- [Getting started with user pools](#)
- [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#)
- [Understanding API, OIDC, and managed login pages authentication](#)

User pool authentication with a third-party identity provider

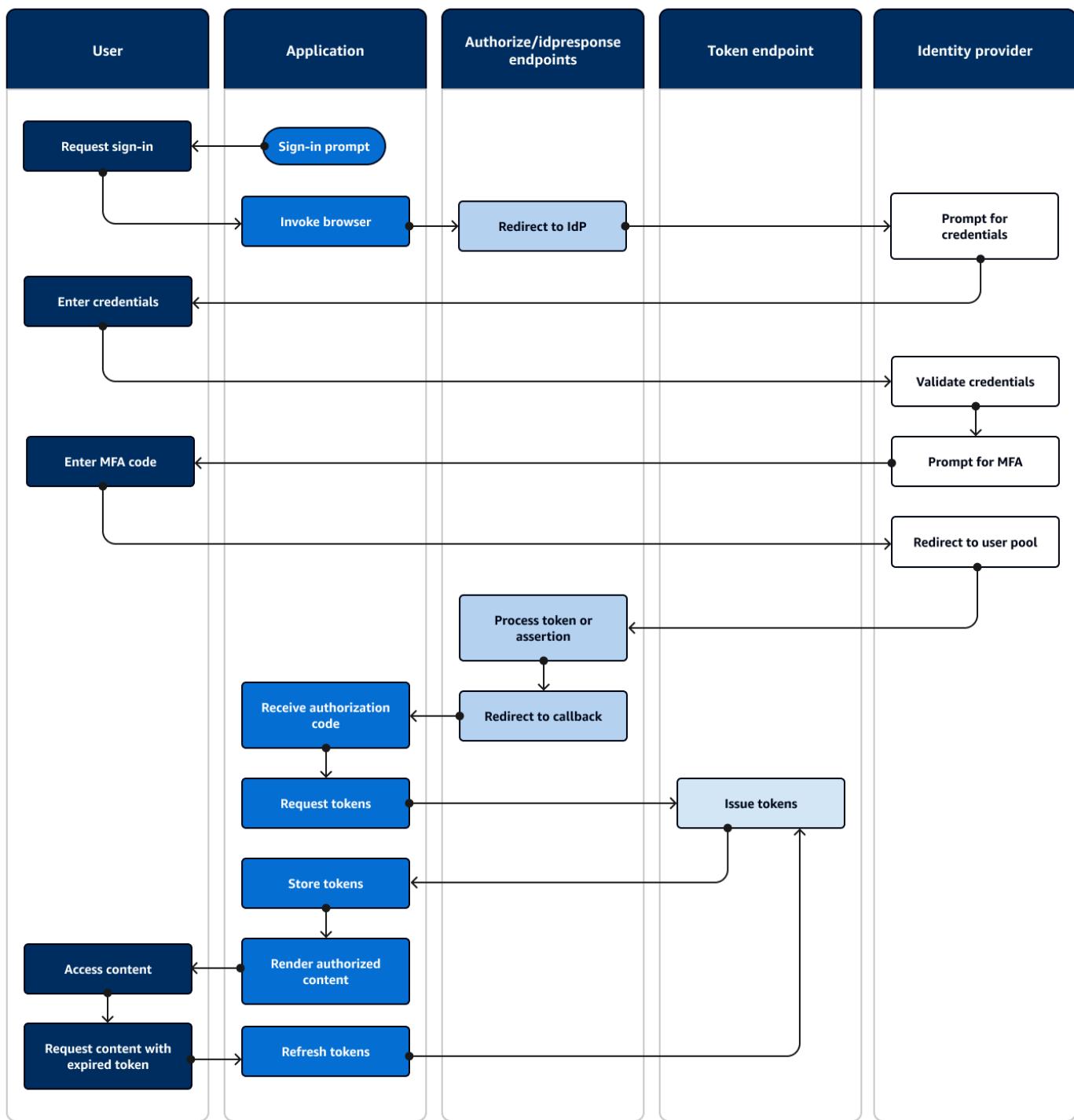
Sign-in with an external identity provider (IdP), or *federated authentication*, is a similar model to [managed login](#). Your application is an OIDC relying party to your user pool, while your user pool serves as a passthrough to an IdP. The IdP can be a consumer user directory like Facebook or Google, or it can be a SAML 2.0 or OIDC enterprise directory like Azure.

Instead of managed login in the user's browser, your application invokes a redirect endpoint on the user pool [authorization server](#). From the user's view, they choose the sign-in button in your application. Then their IdP prompts them to sign in. Like with managed login authentication, an application collects JWTs at a redirect location in the app.

Authentication with a third-party IdP fits a model where users might not want to come up with a new password when they sign up for your application. Third-party authentication can be added with low effort to an application that's implemented managed login authentication. In effect, managed login and third-party IdPs produce a consistent authentication outcome from minor variations in what you invoke in users' browsers.

Like managed login authentication, federated authentication is best for advanced authorization models with OAuth 2.0 scopes.

The following diagram illustrates a typical sign-in session for federated authentication.



Federated authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.

3. The application directs the user to a sign-in prompt with their IdP.
4. They enter their username and password.
5. The IdP validates the user's credentials and determines that the user has activated multi-factor authentication (MFA).
6. The IdP prompts the user to enter an MFA code.
7. The user enters their MFA code.
8. The IdP redirects the user to the user pool with a SAML response or an authorization code.
9. If the user passed an authorization code, the user pool silently exchanges the code for IdP tokens. The user pool validates the IdP tokens and redirects the user to the application with a new authorization code.
10. The application collects the authorization code from the URL request parameter that the user pool appended to the [callback URL](#).
11. The application requests tokens with the authorization code.
12. The token endpoint returns JWTs to the application.
13. The application decodes, validates, and stores or caches the user's JWTs.
14. The application displays the requested access-controlled component.
15. The user views their content.
16. Later, the user's access token has expired, and they request to view an access-controlled component.
17. The application determines that the user's session should persist. It requests new tokens from the token endpoint with the refresh token.

Variants and customization

You can initiate federated authentication in [managed login](#), where users can choose from a list of IdPs that you assigned to your [app client](#). Managed login can also prompt for an email address and [automatically route a user's request](#) to the corresponding SAML IdP. Authentication with a third-party identity provider doesn't *require* user interaction with managed login. Your application can add a request parameter to a user's [authorization server request](#) and cause the user to silently redirect to their IdP sign-in page.

Related resources

- [User pool sign-in with a third party identity provider](#)

- [Scopes, M2M, and APIs with resource servers](#)
- [User pool endpoints and managed login reference](#)

Identity pool authentication

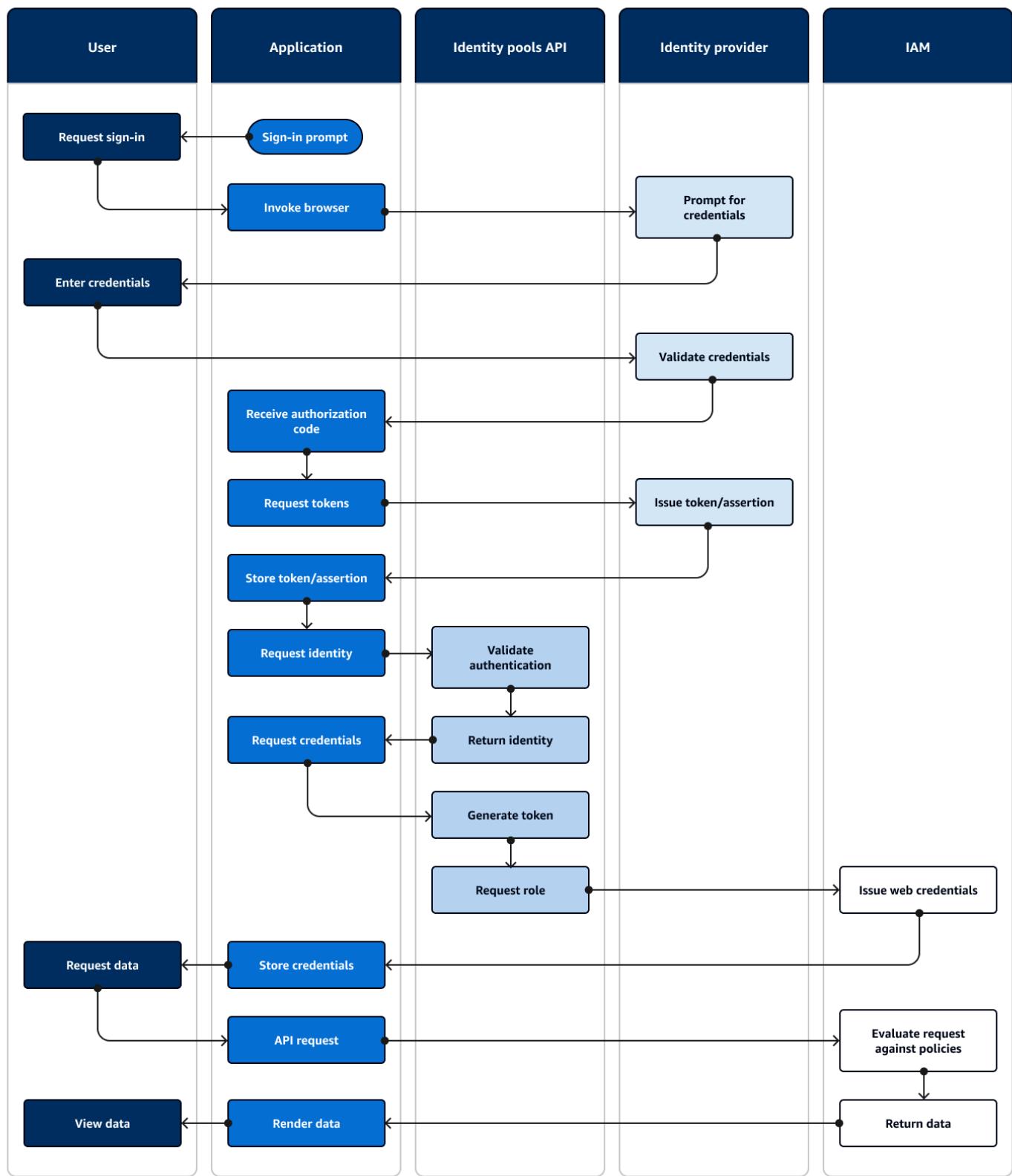
An identity pool is a component for your application that is distinct from a user pool in function, API namespace, and SDK model. Where user pools offer token-based authentication and authorization, identity pools offer authorization for AWS Identity and Access Management (IAM).

You can assign a set of IdPs to identity pools and sign in users with them. User pools are closely integrated as identity pool IdPs and give identity pools the most options for access control. At the same time, there is a wide selection of authentication options for identity pools. User pools join SAML, OIDC, social, developer, and guest identity sources as routes to temporary AWS credentials from identity pools.

Authentication with an identity pool is external—it follows one of the previously illustrated user pool flows, or a flow that you develop independently with another IdP. After your application performs initial authentication, it passes the proof to an identity pool and receives a temporary session in return.

Authentication with an identity pool fits a model where you enforce the access control for application assets and data in AWS services with IAM authorization. Like with [API authentication in user pools](#), a successful application includes AWS SDKs for each of the services that you want to access for your users' benefit. AWS SDKs apply the credentials from identity pool authentication as signatures to API requests.

The following diagram illustrates a typical sign-in session for identity pool authentication with an IdP.



Identity pool authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. The application directs the user to a sign-in prompt with their IdP.
4. They enter their username and password.
5. The IdP validates the user's credentials.
6. The IdP redirects the user to the application with a SAML response or an authorization code.
7. If the user passed an authorization code, the application exchanges the code for IdP tokens.
8. The application decodes, validates, and stores or caches the user's JWTs or assertion.
9. The application invokes the method that makes a [GetId](#) API request. It passes the user's token or assertion and requests an identity ID.
10. The identity pool validates the token or assertion against configured identity providers.
11. The identity pool returns an identity ID.
12. The application invokes the method that makes a [GetCredentialsForIdentity](#) API request. It passes the user's token or assertions and requests an IAM role.
13. The identity pool generates a new JWT. The new JWT contains claims that request an IAM role.
The identity pool determines the role based on the user's request and the role-selection criteria in the identity pool configuration for the IdP.
14. AWS Security Token Service (AWS STS) responds to the [AssumeRoleWithWebIdentity](#) request from the identity pool. The response contains API credentials for a temporary session with an IAM role.
15. The application stores the session credentials.
16. The user takes an action in the app that requires access-protected resources in AWS.
17. The application applies the temporary credentials as [signatures](#) to API requests for the required AWS services.
18. IAM evaluates the policies attached to the role in the credentials. It compares them to the request.
19. The AWS service returns the requested data.
20. The application renders the data in the user's interface.
21. The user views the data.

Variants and customization

To visualize authentication with a user pool, insert one of the previous user-pool overviews after the **Issue token/assertion** step. Developer authentication replaces all steps before **Request identity** with a request signed by [developer credentials](#). Guest authentication also skips straight to **Request identity**, doesn't validate authentication, and returns credentials for a [limited-access IAM role](#).

Related resources

- [Amazon Cognito identity pools](#)
- [User IAM roles](#)
- [Identity pools authentication flow](#)

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	Tools for PowerShell code examples

SDK documentation	Code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

 **Example availability**

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Authorization with Amazon Verified Permissions

[Amazon Verified Permissions](#) is an authorization service for the applications that you build. When you add an Amazon Cognito user pool as an identity source, your app can pass user pool access or identity (ID) tokens to Verified Permissions for an allow or deny decision. Verified Permissions considers your user's properties and request context based on policies that you write in [Cedar Policy Language](#). The request context can include an identifier for the document, image, or other resource they requested, and the action that your user wants to take on the resource.

Your app can provide your user's identity or access tokens to Verified Permissions in [IsAuthorizedWithToken](#) or [BatchIsAuthorizedWithToken](#) API requests. These API operations accept your users as a Principal and make authorization decisions for the Action on the Resource that they want to access. Additional custom Context can contribute to a detailed access decision.

When your app presents a token in an IsAuthorizedWithToken API request, Verified Permissions performs the following validations.

1. Your user pool is a configured Verified Permissions [identity source](#) for the requested policy store.

2. The `client_id` or `aud` claim, in your access or identity token respectively, matches a user pool app client ID that you provided to Verified Permissions. To verify this claim, you must [configure client ID validation](#) in your Verified Permissions identity source.
3. Your token isn't expired.
4. The value of the `token_use` claim in your token matches the parameters that you passed to `IsAuthorizedWithToken`. The `token_use` claim must be `access` if you passed it to the `accessToken` parameter, and `id` if you passed it to the `identityToken` parameter.
5. The signature in your token comes from the published JSON web keys (JWKS) of your user pool. You can view your JWKS at <https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/jwks.json>.

Revoked tokens and deleted users

Verified Permissions only validates the information it knows from your identity source and from the expiration time of your user's token. Verified Permissions doesn't check for token revocation or user existence. If you revoked your user's token or deleted your user's profile from your user pool, Verified Permissions still considers the token valid until it expires.

Policy evaluation

Configure your user pool as an [identity source](#) for your [policy store](#). Configure your app to submit your users' tokens in requests to Verified Permissions. For each request, Verified Permissions compares the claims in the token to a policy. A Verified Permissions policy is like an IAM policy in AWS. It declares a *principal*, *resource*, and *action*. Verified Permissions responds to your request with `Allow` if it matches an allowed action and doesn't match an explicit `Deny` action; otherwise, it responds with `Deny`. For more information, see [Amazon Verified Permissions policies](#) in the [Amazon Verified Permissions User Guide](#).

Customizing tokens

To change, add, and remove the user claims that you want to present to Verified Permissions, customize the content in your access and identity tokens with a [Pre token generation Lambda trigger](#). With a pre token generation trigger, you can add and modify claims in your tokens. For example, you can query a database for additional user attributes and encode them into your ID token.

Note

Because of the way that Verified Permissions processes claims, don't add claims named cognito, dev, or custom in your pre token generation function. When you present these reserved claim prefixes not in colon-delimited format like cognito:username but as full claim names, your authorization requests fail.

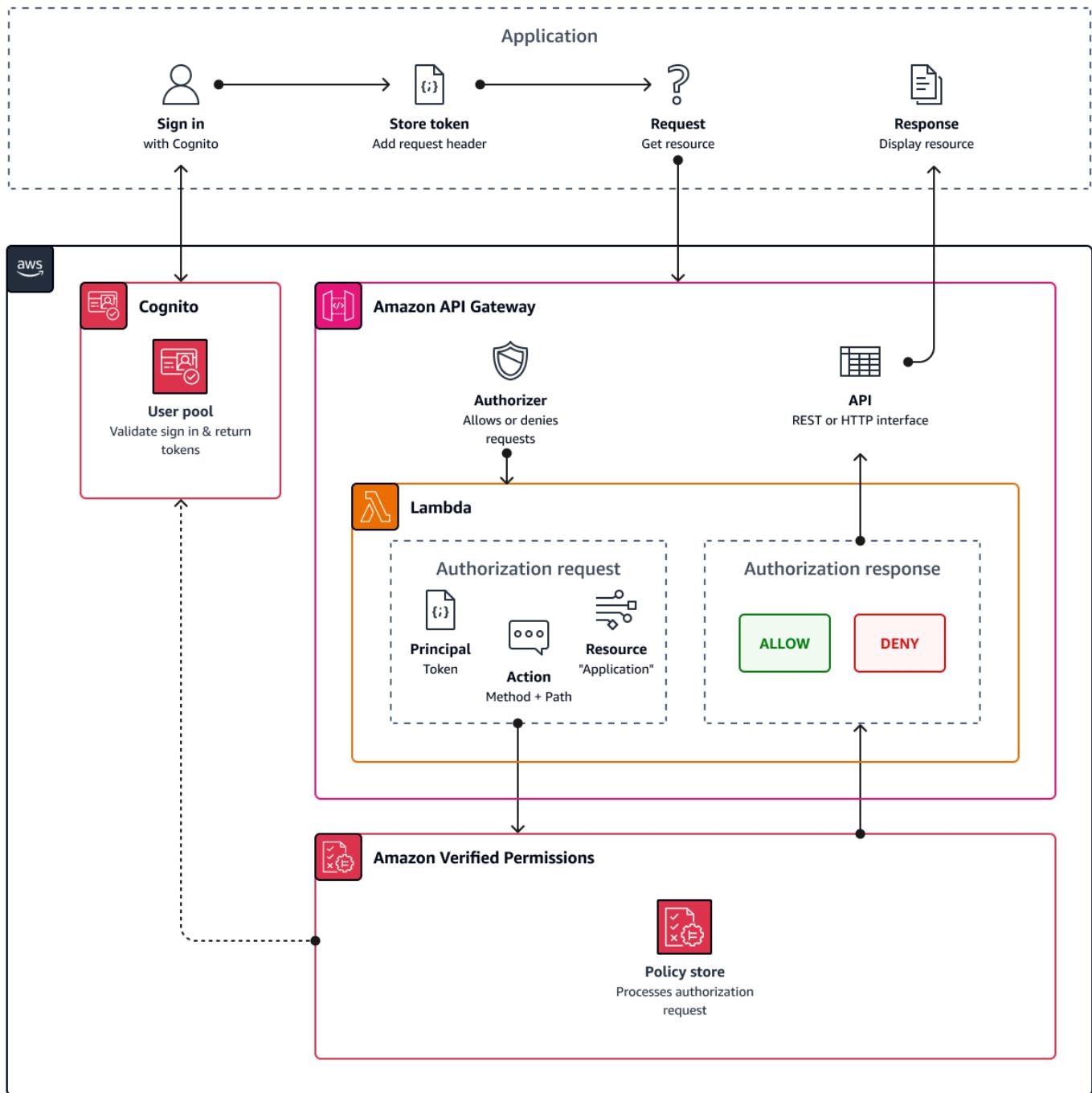
Additional resources

- [Mapping Amazon Cognito tokens to Verified Permissions schema](#)
- [Authorize API Gateway APIs using Amazon Verified Permissions and Amazon Cognito](#)

API authorization with Verified Permissions

Your ID or access tokens can authorize requests to back-end Amazon API Gateway REST APIs with Verified Permissions. You can create a [policy store](#) with immediate links to your user pool and API. With the [Set up with Cognito and API Gateway](#) starting option, Verified Permissions adds a user pool identity source to the policy store, and a Lambda authorizer to the API. When your application passes a user pool bearer token to the API, the Lambda authorizer invokes Verified Permissions. The authorizer passes the token as a principal and the request path and method as an action.

The following diagram illustrates the authorization flow for an API Gateway API with Verified Permissions. For a detailed breakdown, see [API-linked policy stores](#) in the Amazon Verified Permissions User Guide.



Verified Permissions structures API authorization around [user pool groups](#). Because both ID and access tokens include a `cognito:groups` claim, your policy store can manage role-based access control (RBAC) for your APIs in a variety of application contexts.

Choosing policy store settings

When you configure an identity source on a policy store, you must choose whether you want to process access or ID tokens. This decision is significant to the way that your policy engine operates. ID tokens contain user attributes. Access tokens contain user access-control information: [OAuth scopes](#). Although both token types have group-membership information, we generally recommend the access token for RBAC with a Verified Permissions policy store. The access token adds to group membership with scopes that can contribute to the authorization decision. The claims in an access token become [context](#) in the authorization request.

You must also configure the user and group entity types when you configure a user pool as an identity source. Entity types are principal, action, and resource identifiers that you can reference in Verified Permissions policies. Entities in policy stores can have a *membership* relationship, where one entity can be a member of a *parent* entity. With membership, you can reference principal groups, action groups, and resource groups. In the case of user pool groups, the user entity type that you specify must be a member of the group entity type. When you set up an [API-linked policy store](#) or follow **Guided setup** in the Verified Permissions console, your policy store automatically has this parent-member relationship.

The ID token can combine RBAC with attribute-based access control (ABAC). After you create an [API-linked policy store](#), you can enhance your policies with [user attributes](#) and group membership. The attribute claims in an ID token become [principal attributes](#) in the authorization request. Your policies can make authorization decisions based on principal attributes.

You can also configure a policy store to accept tokens with an aud or `client_id` claim that matches a list of acceptable app clients that you provide.

Example policy for role-based API authorization

The following example policy was created by the setup of a Verified Permissions policy store for a [PetStore](#) example REST API.

```
permit(
    principal in PetStore::UserGroup::"us-east-1_EXAMPLE|MyGroup",
    action in [ PetStore::Action::"get /pets", PetStore::Action::"get /pets/{petId}" ],
    resource
);
```

Verified Permissions returns an Allow decision to the authorization request from your application when:

1. Your application passed an ID or access token in an Authorization header as a bearer token.
2. Your application passed a token with a cognito:groups claim that contains the string MyGroup.
3. Your application made an HTTP GET request to, for example, <https://myapi.example.com/pets> or <https://myapi.example.com/pets/scrappy>.

Example policy for an Amazon Cognito user

Your user pool can also generate authorization requests to Verified Permissions in conditions other than API requests. You can submit any access control decisions in your application to your policy store. For example, you can supplement Amazon DynamoDB or Amazon S3 security with attribute-based access control before any requests transit the network, reducing quota use.

The following example uses the [Cedar Policy Language](#) to permit Finance users who authenticate with one user pool app client to read and write example_image.png. John, a user in your app, receives an ID token from your app client and passes it in a GET request to a URL that requires authorization, https://example.com/images/example_image.png. John's ID token has an aud claim of your user pool app client ID 1234567890example. Your pre token generation Lambda function also inserted a new claim costCenter with a value, for John, of Finance1234.

```
permit (
    principal,
    actions in [ExampleCorp::Action::"readFile", "writeFile"],
    resource == ExampleCorp::Photo::"example_image.png"
)
when {
    principal.aud == "1234567890example" &&
    principal.custom.costCenter like "Finance*"
};
```

The following request body results in an Allow response.

```
{
    "accesstoken": "[John's ID token]",
    "action": {
        "actionId": "readFile",
        "actionType": "Action"
    },
}
```

```
"resource": {  
    "entityId": "example_image.png",  
    "entityType": "Photo"  
}  
}
```

When you want to specify a principal in a Verified Permissions policy, use the following format:

```
permit (  
    principal == [Namespace]::[Entity]::"[user pool ID]"|[user sub]",  
    action,  
    resource  
);
```

The following is an example principal for a user in a user pool with ID us-east-1_Example with sub, or user ID, 973db890-092c-49e4-a9d0-912a4c0a20c7.

```
principal == ExampleCorp::User::"us-east-1_Example|973db890-092c-49e4-  
a9d0-912a4c0a20c7",
```

When you want to specify a user group in a Verified Permissions policy, use the following format:

```
permit (  
    principal in [Namespace]::[Group Entity]::"[Group name]",  
    action,  
    resource  
);
```

The following is an example

Attribute-based access control

Authorization with Verified Permissions for your apps, and the [attributes for access control](#) feature of Amazon Cognito identity pools for AWS credentials, are both forms of attribute-based access control (ABAC). The following is a comparison of the features of Verified Permissions and Amazon Cognito ABAC. In ABAC, a system examines the attributes of an entity and makes an authorization decision from conditions that you define.

Service	Process	Result
Amazon Verified Permissions	Returns an Allow or Deny decision from analysis of a user pool JWT.	Access to application resources succeeds or fails based on Cedar policy evaluation.
Amazon Cognito identity pools (attributes for access control)	Assigns session tags to your user based on their attributes. IAM policy conditions can check tags. Allow or Deny user access to AWS services.	A tagged session with temporary AWS credentials for an IAM role.

Code examples for Amazon Cognito using AWS SDKs

The following code examples show how to use Amazon Cognito with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Code examples for Amazon Cognito Identity using AWS SDKs](#)
 - [Basic examples for Amazon Cognito Identity using AWS SDKs](#)
 - [Actions for Amazon Cognito Identity using AWS SDKs](#)
 - [Use CreateIdentityPool with an AWS SDK or CLI](#)
 - [Use DeleteIdentityPool with an AWS SDK or CLI](#)
 - [Use DescribeIdentityPool with a CLI](#)
 - [Use GetCredentialsForIdentity with an AWS SDK](#)
 - [Use GetIdentityPoolRoles with a CLI](#)
 - [Use ListIdentityPools with an AWS SDK or CLI](#)
 - [Use SetIdentityPoolRoles with a CLI](#)
 - [Use UpdateIdentityPool with a CLI](#)
 - [Scenarios for Amazon Cognito Identity using AWS SDKs](#)
 - [Create an Amazon Textract explorer application](#)
 - [Code examples for Amazon Cognito Identity Provider using AWS SDKs](#)
 - [Basic examples for Amazon Cognito Identity Provider using AWS SDKs](#)
 - [Hello Amazon Cognito](#)
 - [Actions for Amazon Cognito Identity Provider using AWS SDKs](#)
 - [Use AdminCreateUser with an AWS SDK or CLI](#)
 - [Use Admin GetUser with an AWS SDK or CLI](#)
 - [Use AdminInitiateAuth with an AWS SDK or CLI](#)
 - [Use AdminRespondToAuthChallenge with an AWS SDK](#)
 - [Use AdminSetUserPassword with an AWS SDK](#)

- [Use AssociateSoftwareToken with an AWS SDK](#)
- [Use ConfirmDevice with an AWS SDK](#)
- [Use ConfirmForgotPassword with an AWS SDK or CLI](#)
- [Use ConfirmSignUp with an AWS SDK or CLI](#)
- [Use CreateUserPool with an AWS SDK or CLI](#)
- [Use CreateUserPoolClient with an AWS SDK or CLI](#)
- [Use DeleteUser with an AWS SDK or CLI](#)
- [Use ForgotPassword with an AWS SDK or CLI](#)
- [Use InitiateAuth with an AWS SDK](#)
- [Use ListUserPools with an AWS SDK or CLI](#)
- [Use ListUsers with an AWS SDK or CLI](#)
- [Use ResendConfirmationCode with an AWS SDK or CLI](#)
- [Use RespondToAuthChallenge with an AWS SDK or CLI](#)
- [Use SignUp with an AWS SDK or CLI](#)
- [Use UpdateUserPool with an AWS SDK or CLI](#)
- [Use VerifySoftwareToken with an AWS SDK](#)
- [Scenarios for Amazon Cognito Identity Provider using AWS SDKs](#)
 - [Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK](#)
 - [Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK](#)
 - [Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK](#)
 - [Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK](#)
- [Code examples for Amazon Cognito Sync using AWS SDKs](#)
 - [Basic examples for Amazon Cognito Sync using AWS SDKs](#)
 - [Actions for Amazon Cognito Sync using AWS SDKs](#)
 - [Use ListIdentityPoolUsage with an AWS SDK](#)

Code examples for Amazon Cognito Identity using AWS SDKs

The following code examples show how to use Amazon Cognito Identity with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Basic examples for Amazon Cognito Identity using AWS SDKs](#)
 - [Actions for Amazon Cognito Identity using AWS SDKs](#)
 - [Use CreateIdentityPool with an AWS SDK or CLI](#)
 - [Use DeleteIdentityPool with an AWS SDK or CLI](#)
 - [Use DescribeIdentityPool with a CLI](#)
 - [Use GetCredentialsForIdentity with an AWS SDK](#)
 - [Use GetIdentityPoolRoles with a CLI](#)
 - [Use ListIdentityPools with an AWS SDK or CLI](#)
 - [Use SetIdentityPoolRoles with a CLI](#)
 - [Use UpdateIdentityPool with a CLI](#)
 - [Scenarios for Amazon Cognito Identity using AWS SDKs](#)
 - [Create an Amazon Textract explorer application](#)

Basic examples for Amazon Cognito Identity using AWS SDKs

The following code examples show how to use the basics of Amazon Cognito Identity with AWS SDKs.

Examples

- [Actions for Amazon Cognito Identity using AWS SDKs](#)

- [Use CreateIdentityPool with an AWS SDK or CLI](#)
- [Use DeleteIdentityPool with an AWS SDK or CLI](#)
- [Use DescribeIdentityPool with a CLI](#)
- [Use GetCredentialsForIdentity with an AWS SDK](#)
- [Use GetIdentityPoolRoles with a CLI](#)
- [Use ListIdentityPools with an AWS SDK or CLI](#)
- [Use SetIdentityPoolRoles with a CLI](#)
- [Use UpdateIdentityPool with a CLI](#)

Actions for Amazon Cognito Identity using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Cognito Identity actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon Cognito Identity API and are code excerpts from larger programs that must be run in context. You can see actions in context in [Scenarios for Amazon Cognito Identity using AWS SDKs](#).

The following examples include only the most commonly used actions. For a complete list, see the [Amazon Cognito Identity API Reference](#).

Examples

- [Use CreateIdentityPool with an AWS SDK or CLI](#)
- [Use DeleteIdentityPool with an AWS SDK or CLI](#)
- [Use DescribeIdentityPool with a CLI](#)
- [Use GetCredentialsForIdentity with an AWS SDK](#)
- [Use GetIdentityPoolRoles with a CLI](#)
- [Use ListIdentityPools with an AWS SDK or CLI](#)
- [Use SetIdentityPoolRoles with a CLI](#)
- [Use UpdateIdentityPool with a CLI](#)

Use CreateIdentityPool with an AWS SDK or CLI

The following code examples show how to use `CreateIdentityPool`.

CLI

AWS CLI

To create an identity pool with Cognito identity pool provider

This example creates an identity pool named MyIdentityPool. It has a Cognito identity pool provider. Unauthenticated identities are not allowed.

Command:

```
aws cognito-identity create-identity-pool --identity-pool-name MyIdentityPool --no-allow-unauthenticated-identities --cognito-identity-providers ProviderName="cognito-idp.us-west-2.amazonaws.com/us-west-2_aaaaaaaaaa",ClientId="3n4b5urk1ft4fl3mg5e62d9ado",ServerSideTokenCheck=false
```

Output:

```
{  
    "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",  
    "IdentityPoolName": "MyIdentityPool",  
    "AllowUnauthenticatedIdentities": false,  
    "CognitoIdentityProviders": [  
        {  
            "ProviderName": "cognito-idp.us-west-2.amazonaws.com/us-west-2_1111111111",  
            "ClientId": "3n4b5urk1ft4fl3mg5e62d9ado",  
            "ServerSideTokenCheck": false  
        }  
    ]  
}
```

- For API details, see [CreateIdentityPool](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
software.amazon.awssdk.services.cognitoidentity.model.CreateIdentityPoolRequest;
import
software.amazon.awssdk.services.cognitoidentity.model.CreateIdentityPoolResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CreateIdentityPool {
    public static void main(String[] args) {
        final String usage = """
            Usage:
            <identityPoolName>\s

            Where:
            identityPoolName - The name to give your identity pool.
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String identityPoolName = args[0];
CognitoIdentityClient cognitoClient = CognitoIdentityClient.builder()
    .region(Region.US_EAST_1)
    .build();

String identityPoolId = createIdPool(cognitoClient, identityPoolName);
System.out.println("Unity pool ID " + identityPoolId);
cognitoClient.close();
}

public static String createIdPool(CognitoIdentityClient cognitoClient, String
identityPoolName) {
    try {
        CreateIdentityPoolRequest poolRequest =
CreateIdentityPoolRequest.builder()
            .allowUnauthenticatedIdentities(false)
            .identityPoolName(identityPoolName)
            .build();

        CreateIdentityPoolResponse response =
cognitoClient.createIdentityPool(poolRequest);
        return response.identityPoolId();

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

- For API details, see [CreateIdentityPool](#) in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell

Example 1: Creates a new Identity Pool which allows unauthenticated identities.

```
New-CGIIIdentityPool -AllowUnauthenticatedIdentities $true -IdentityPoolName CommonTests13
```

Output:

```
LoggedAt          : 8/12/2015 4:56:07 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName   :
IdentityPoolId      : us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3
IdentityPoolName    : CommonTests13
OpenIdConnectProviderARNs : {}
SupportedLoginProviders : {}
ResponseMetadata     : Amazon.Runtime.ResponseMetadata
ContentLength        : 136
HttpStatusCode       : OK
```

- For API details, see [CreateIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import AWSCognitoIdentity

/// Create a new identity pool and return its ID.
///
/// - Parameters:
///   - name: The name to give the new identity pool.
///
/// - Returns: A string containing the newly created pool's ID, or `nil` if an error occurred.
///
func createIdentityPool(name: String) async throws -> String? {
```

```
do {
    let cognitoInputCall = CreateIdentityPoolInput(developerProviderName:
"com.exampleco.CognitoIdentityDemo",
                                              identityPoolName:
name)

    let result = try await
cognitoIdentityClient.createIdentityPool(input: cognitoInputCall)
    guard let poolId = result.identityPoolId else {
        return nil
    }

    return poolId
} catch {
    print("ERROR: createIdentityPool:", dump(error))
    throw error
}
}
```

- For more information, see [AWS SDK for Swift developer guide](#).
- For API details, see [CreateIdentityPool in AWS SDK for Swift API reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteIdentityPool` with an AWS SDK or CLI

The following code examples show how to use `DeleteIdentityPool`.

CLI

AWS CLI

To delete identity pool

The following `delete-identity-pool` example deletes the specified identity pool.

Command:

```
aws cognito-identity delete-identity-pool \
```

```
--identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111"
```

This command produces no output.

- For API details, see [DeleteIdentityPool](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.awscore.exception.AwsServiceException;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
software.amazon.awssdk.services.cognitoidentity.model.DeleteIdentityPoolRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class DeleteIdentityPool {

    public static void main(String[] args) {
        final String usage = """

            Usage:
            <identityPoolId>\s

            Where:
            identityPoolId - The Id value of your identity pool.

    
```

```
""";  
  
    if (args.length != 1) {  
        System.out.println(usage);  
        System.exit(1);  
    }  
  
    String identityPoold = args[0];  
    CognitoIdentityClient cognitoIdClient = CognitoIdentityClient.builder()  
        .region(Region.US_EAST_1)  
        .credentialsProvider(ProfileCredentialsProvider.create())  
        .build();  
  
    deleteIdPool(cognitoIdClient, identityPoold);  
    cognitoIdClient.close();  
}  
  
public static void deleteIdPool(CognitoIdentityClient cognitoIdClient, String  
identityPoold) {  
    try {  
  
        DeleteIdentityPoolRequest identityPoolRequest =  
DeleteIdentityPoolRequest.builder()  
            .identityPoolId(identityPoold)  
            .build();  
  
        cognitoIdClient.deleteIdentityPool(identityPoolRequest);  
        System.out.println("Done");  
  
    } catch (AwsServiceException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}  
}
```

- For API details, see [DeleteIdentityPool](#) in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell

Example 1: Deletes a specific Identity Pool.

```
Remove-CGIIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-  
EXAMPLEGUID1
```

- For API details, see [DeleteIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import AWSIdentity
```



```
/// Delete the specified identity pool.  
///  
/// - Parameters:  
///   - id: The ID of the identity pool to delete.  
///  
func deleteIdentityPool(id: String) async throws {  
    do {  
        let input = DeleteIdentityPoolInput(  
            identityPoolId: id  
        )  
  
        _ = try await cognitoIdentityClient.deleteIdentityPool(input: input)  
    } catch {  
        print("ERROR: deleteIdentityPool:", dump(error))  
        throw error  
    }  
}
```

- For more information, see [AWS SDK for Swift developer guide](#).
- For API details, see [DeleteIdentityPool](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeIdentityPool` with a CLI

The following code examples show how to use `DescribeIdentityPool`.

CLI

AWS CLI

To describe an identity pool

This example describes an identity pool.

Command:

```
aws cognito-identity describe-identity-pool --identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111"
```

Output:

```
{  
    "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",  
    "IdentityPoolName": "MyIdentityPool",  
    "AllowUnauthenticatedIdentities": false,  
    "CognitoIdentityProviders": [  
        {  
            "ProviderName": "cognito-idp.us-west-2.amazonaws.com/us-west-2_11111111",  
            "ClientId": "3n4b5urk1ft4fl3mg5e62d9ado",  
            "ServerSideTokenCheck": false  
        }  
    ]}
```

{}

- For API details, see [DescribeIdentityPool](#) in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: Retrieves information about a specific Identity Pool by its id.

```
Get-CGIIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-  
EXAMPLEGUID1
```

Output:

```
LoggedAt          : 8/12/2015 4:29:40 PM  
AllowUnauthenticatedIdentities : True  
DeveloperProviderName      :  
IdentityPoolId           : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1  
IdentityPoolName         : CommonTests1  
OpenIdConnectProviderARNs : {}  
SupportedLoginProviders   : {}  
ResponseMetadata        : Amazon.Runtime.ResponseMetadata  
ContentLength           : 142  
HttpStatusCode          : OK
```

- For API details, see [DescribeIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetCredentialsForIdentity with an AWS SDK

The following code example shows how to use GetCredentialsForIdentity.

Java

SDK for Java 2.x**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
software.amazon.awssdk.services.cognitoidentity.model.GetCredentialsForIdentityRequest;
import
software.amazon.awssdk.services.cognitoidentity.model.GetCredentialsForIdentityResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class GetIdentityCredentials {
    public static void main(String[] args) {

        final String usage = """
            Usage:
            <identityId>\s

            Where:
            identityId - The Id of an existing identity in the format
REGION:GUID.
            """;

        if (args.length != 1) {
```

```
        System.out.println(usage);
        System.exit(1);
    }

    String identityId = args[0];
    CognitoIdentityClient cognitoClient = CognitoIdentityClient.builder()
        .region(Region.US_EAST_1)
        .build();

    getCredsForIdentity(cognitoClient, identityId);
    cognitoClient.close();
}

public static void getCredsForIdentity(CognitoIdentityClient cognitoClient,
String identityId) {
    try {
        GetCredentialsForIdentityRequest getCredentialsForIdentityRequest =
GetCredentialsForIdentityRequest
            .builder()
            .identityId(identityId)
            .build();

        GetCredentialsForIdentityResponse response = cognitoClient
            .getCredentialsForIdentity(getCredentialsForIdentityRequest);
        System.out.println(
            "Identity ID " + response.identityId() + ", Access key ID " +
response.credentials().accessKeyId());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [GetCredentialsForIdentity](#) in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetIdentityPoolRoles with a CLI

The following code examples show how to use GetIdentityPoolRoles.

CLI

AWS CLI

To get identity pool roles

This example gets identity pool roles.

Command:

```
aws cognito-identity get-identity-pool-roles --identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111"
```

Output:

```
{  
    "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",  
    "Roles": {  
        "authenticated": "arn:aws:iam::111111111111:role/Cognito_MyIdentityPoolAuth_Role",  
        "unauthenticated": "arn:aws:iam::111111111111:role/Cognito_MyIdentityPoolUnauth_Role"  
    }  
}
```

- For API details, see [GetIdentityPoolRoles](#) in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: Gets the information about roles for a specific Identity Pool.

```
Get-CGIIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
```

Output:

```
LoggedAt      : 8/12/2015 4:33:51 PM
IdentityPoolId : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
Roles         : {[unauthenticated, arn:aws:iam::123456789012:role/
CommonTests1Role]}
ResponseMetadata : Amazon.Runtime.ResponseMetadata
ContentLength   : 165
HttpStatusCode : OK
```

- For API details, see [GetIdentityPoolRoles](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ListIdentityPools` with an AWS SDK or CLI

The following code examples show how to use `ListIdentityPools`.

CLI

AWS CLI

To list identity pools

This example lists identity pools. There s a maximum of 20 identities listed.

Command:

```
aws cognito-identity list-identity-pools --max-results 20
```

Output:

```
{
    "IdentityPools": [
        {
            "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
            "IdentityPoolName": "MyIdentityPool"
        },
        {
            "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
            "IdentityPoolName": "AnotherIdentityPool"
        }
    ]
}
```

```
        },
        {
            "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
            "IdentityPoolName": "IdentityPoolRegionA"
        }
    ]
}
```

- For API details, see [ListIdentityPools](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
software.amazon.awssdk.services.cognitoidentity.model.ListIdentityPoolsRequest;
import
software.amazon.awssdk.services.cognitoidentity.model.ListIdentityPoolsResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListIdentityPools {
    public static void main(String[] args) {
        CognitoIdentityClient cognitoClient = CognitoIdentityClient.builder()
            .region(Region.US_EAST_1)
```

```
        .build();

    listIdPools(cognitoClient);
    cognitoClient.close();
}

public static void listIdPools(CognitoIdentityClient cognitoClient) {
    try {
        ListIdentityPoolsRequest poolsRequest =
ListIdentityPoolsRequest.builder()
            .maxResults(15)
            .build();

        ListIdentityPoolsResponse response =
cognitoClient.listIdentityPools(poolsRequest);
        response.identityPools().forEach(pool -> {
            System.out.println("Pool ID: " + pool.identityPoolId());
            System.out.println("Pool name: " + pool.identityPoolName());
        });
    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [ListIdentityPools](#) in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell

Example 1: Retrieves a list of existing Identity Pools.

```
Get-CGIIIdentityPoolList
```

Output:

```
IdentityPoolId
IdentityPoolName
```


us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1	CommonTests1
us-east-1:118d242d-204e-4b88-b803-EXAMPLEGUID2	Tests2
us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3	CommonTests13

- For API details, see [ListIdentityPools](#) in *AWS Tools for PowerShell Cmdlet Reference*.

Swift

SDK for Swift

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import AWSCognitoIdentity

/// Return the ID of the identity pool with the specified name.
///
/// - Parameters:
///   - name: The name of the identity pool whose ID should be returned.
///
/// - Returns: A string containing the ID of the specified identity pool
/// or `nil` on error or if not found.
///
func getIdentityPoolID(name: String) async throws -> String? {
    let listPoolsInput = ListIdentityPoolsInput(maxResults: 25)
    // Use "Paginated" to get all the objects.
    // This lets the SDK handle the 'nextToken' field in
    "ListIdentityPoolsOutput".
    let pages = cognitoIdentityClient.listIdentityPoolsPaginated(input:
listPoolsInput)

    do {
        for try await page in pages {
            guard let identityPools = page.identityPools else {
                print("ERROR: listIdentityPoolsPaginated returned nil
contents.")
            }
        }
    }
}
```

```
        continue
    }

    /// Read pages of identity pools from Cognito until one is found
    /// whose name matches the one specified in the `name` parameter.
    /// Return the matching pool's ID.

    for pool in identityPools {
        if pool.identityPoolName == name {
            return pool.identityPoolId!
        }
    }
}

} catch {
    print("ERROR: getIdentityPoolID:", dump(error))
    throw error
}

return nil
}
```

Get the ID of an existing identity pool or create it if it doesn't already exist.

```
import AWSCognitoIdentity

/// Return the ID of the identity pool with the specified name.
///
/// - Parameters:
///   - name: The name of the identity pool whose ID should be returned
///
/// - Returns: A string containing the ID of the specified identity pool.
/// Returns `nil` if there's an error or if the pool isn't found.
///
public func getOrCreateIdentityPoolID(name: String) async throws -> String? {
    // See if the pool already exists. If it doesn't, create it.

    do {
        guard let poolId = try await getIdentityPoolID(name: name) else {
            return try await createIdentityPool(name: name)
        }
    }
}
```

```
        return poolId
    } catch {
        print("ERROR: getOrCreateIdentityPoolID:", dump(error))
        throw error
    }
}
```

- For more information, see [AWS SDK for Swift developer guide](#).
- For API details, see [ListIdentityPools](#) in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use SetIdentityPoolRoles with a CLI

The following code examples show how to use SetIdentityPoolRoles.

CLI

AWS CLI

To set identity pool roles

The following set-identity-pool-roles example sets an identity pool role.

```
aws cognito-identity set-identity-pool-roles \
    --identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111" \
    --roles authenticated="arn:aws:iam::111111111111:role/
Cognito_MyIdentityPoolAuth_Role"
```

- For API details, see [SetIdentityPoolRoles](#) in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: Configures the specific Identity Pool to have an unauthenticated IAM role.

```
Set-CGIIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-  
EXAMPLEGUID1 -Role @{ "unauthenticated" = "arn:aws:iam::123456789012:role/  
CommonTests1Role" }
```

- For API details, see [SetIdentityPoolRoles](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateIdentityPool` with a CLI

The following code examples show how to use `UpdateIdentityPool`.

CLI

AWS CLI

To update an identity pool

This example updates an identity pool. It sets the name to `MyIdentityPool`. It adds Cognito as an identity provider. It disallows unauthenticated identities.

Command:

```
aws cognito-identity update-identity-pool --identity-pool-id "us-  
west-2:11111111-1111-1111-1111-111111111111" --identity-pool-  
name "MyIdentityPool" --no-allow-unauthenticated-identities --cognito-  
identity-providers ProviderName="cognito-idp.us-west-2.amazonaws.com/us-  
west-2_11111111",ClientId="3n4b5urk1ft4f13mg5e62d9ado",ServerSideTokenCheck=false
```

Output:

```
{  
    "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",  
    "IdentityPoolName": "MyIdentityPool",  
    "AllowUnauthenticatedIdentities": false,  
    "CognitoIdentityProviders": [  
        {  
            "ProviderName": "cognito-idp.us-west-2.amazonaws.com/us-  
west-2_11111111",  
            "ClientId": "3n4b5urk1ft4f13mg5e62d9ado",  
            "ServerSideTokenCheck": false  
        }  
    ]  
}
```

```
        "ServerSideTokenCheck": false
    }
]
}
```

- For API details, see [UpdateIdentityPool](#) in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: Updates some of the Identity Pool properties, in this case the name of the Identity Pool.

```
Update-CGIIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1 -IdentityPoolName NewPoolName
```

Output:

```
LoggedAt          : 8/12/2015 4:53:33 PM
AllowUnauthenticatedIdentities : False
DeveloperProviderName      :
IdentityPoolId           : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
IdentityPoolName          : NewPoolName
OpenIdConnectProviderARNs   : {}
SupportedLoginProviders    : {}
ResponseMetadata          : Amazon.Runtime.ResponseMetadata
ContentLength              : 135
HttpStatusCode             : OK
```

- For API details, see [UpdateIdentityPool](#) in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Cognito Identity using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Cognito Identity with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling

multiple functions within Amazon Cognito Identity or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- [Create an Amazon Textract explorer application](#)

Create an Amazon Textract explorer application

The following code examples show how to explore Amazon Textract output through an interactive application.

JavaScript

SDK for JavaScript (v3)

Shows how to use the AWS SDK for JavaScript to build a React application that uses Amazon Textract to extract data from a document image and display it in an interactive web page. This example runs in a web browser and requires an authenticated Amazon Cognito identity for credentials. It uses Amazon Simple Storage Service (Amazon S3) for storage, and for notifications it polls an Amazon Simple Queue Service (Amazon SQS) queue that is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples for Amazon Cognito Identity Provider using AWS SDKs

The following code examples show how to use Amazon Cognito Identity Provider with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello Amazon Cognito

The following code examples show how to get started using Amazon Cognito.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
```

```
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS cognito-idp)

# Set this project's name.
project("hello_cognito")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
"${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
    # for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
    # may need to uncomment this
    # and set the proper subdirectory to the
    # executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
    hello_cognito.cpp)

target_link_libraries(${PROJECT_NAME}
    ${AWSSDK_LINK_LIBRARIES})
```

Code for the `hello_cognito.cpp` source file.

```
#include <aws/core/Aws.h>
#include <aws/cognito-idp/CognitoIdentityProviderClient.h>
#include <aws/cognito-idp/model/ListUserPoolsRequest.h>
#include <iostream>

/*
 * A "Hello Cognito" starter application which initializes an Amazon Cognito
 * client and lists the Amazon Cognito
 * user pools.
 *
 * main function
 *
 * Usage: 'hello_cognito'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//    options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
cognitoClient(clientConfig);

        Aws::String nextToken; // Used for pagination.
        std::vector<Aws::String> userPools;

        do {
            Aws::CognitoIdentityProvider::Model::ListUserPoolsRequest
listUserPoolsRequest;
            if (!nextToken.empty()) {
                listUserPoolsRequest.SetNextToken(nextToken);
            }
        }
```

```
Aws::CognitoIdentityProvider::Model::ListUserPoolsOutcome
listUserPoolsOutcome =
    cognitoClient.ListUserPools(listUserPoolsRequest);

    if (listUserPoolsOutcome.IsSuccess()) {
        for (auto &userPool:
listUserPoolsOutcome.GetResult(). GetUserPools() {

            userPools.push_back(userPool.GetName());
        }

        nextToken = listUserPoolsOutcome.GetResult().GetNextToken();
    } else {
        std::cerr << "ListUserPools error: " <<
listUserPoolsOutcome.GetError().GetMessage() << std::endl;
        result = 1;
        break;
    }

} while (!nextToken.empty());
std::cout << userPools.size() << " user pools found." << std::endl;
for (auto &userPool: userPools) {
    std::cout << "    user pool: " << userPool << std::endl;
}
}

Aws::ShutdownAPI(options); // Should only be called once.
return result;
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
// Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)
    fmt.Println("Let's list the user pools for your account.")
    var pools []types.UserPoolDescriptionType
    paginator := cognitoidentityprovider.NewListUserPoolsPaginator(
```

```
cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:  
aws.Int32(10)})  
for paginator.HasMorePages() {  
    output, err := paginator.NextPage(ctx)  
    if err != nil {  
        log.Printf("Couldn't get user pools. Here's why: %v\n", err)  
    } else {  
        pools = append(pools, output.UserPools...)  
    }  
}  
if len(pools) == 0 {  
    fmt.Println("You don't have any user pools!")  
} else {  
    for _, pool := range pools {  
        fmt.Printf("\t%v: %v\n", *pool.Name, *pool.Id)  
    }  
}  
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;  
import  
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;  
import  
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc  
import  
    software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsResponse;  
import  
    software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsRequest;
```

```
/**  
 * Before running this Java V2 code example, set up your development  
 * environment, including your credentials.  
 *  
 * For more information, see the following documentation topic:  
 *  
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html  
 */  
public class ListUserPools {  
    public static void main(String[] args) {  
        CognitoIdentityProviderClient cognitoClient =  
CognitoIdentityProviderClient.builder()  
            .region(Region.US_EAST_1)  
            .build();  
  
        listAllUserPools(cognitoClient);  
        cognitoClient.close();  
    }  
  
    public static void listAllUserPools(CognitoIdentityProviderClient  
cognitoClient) {  
        try {  
            ListUserPoolsRequest request = ListUserPoolsRequest.builder()  
                .maxResults(10)  
                .build();  
  
            ListUserPoolsResponse response =  
cognitoClient.listUserPools(request);  
            response.userPools().forEach(userpool -> {  
                System.out.println("User pool " + userpool.name() + ", User ID "  
+ userpool.id());  
            });  
  
        } catch (CognitoIdentityProviderException e) {  
            System.err.println(e.awsErrorDetails().errorMessage());  
            System.exit(1);  
        }  
    }  
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- For API details, see [ListUserPools](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import boto3

# Create a Cognito Identity Provider client
cognitoidp = boto3.client("cognito-idp")

# Initialize a paginator for the list_user_pools operation
paginator = cognitoidp.getPaginator("list_user_pools")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(MaxResults=10)

# Initialize variables for pagination
user_pools = []

# Handle pagination
for page in page_iterator:
    user_pools.extend(page.get("UserPools", []))

# Print the list of user pools
print("User Pools for the account:")
if user_pools:
    for pool in user_pools:
        print(f"Name: {pool['Name']}, ID: {pool['Id']}")
else:
    print("No user pools found.")
```

- For API details, see [ListUserPools](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require 'aws-sdk-cognitoidentityprovider'
require 'logger'

# CognitoManager is a class responsible for managing AWS Cognito operations
# such as listing all user pools in the current AWS account.
class CognitoManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all user pools associated with the AWS account.
  def list_user_pools
    paginator = @client.list_user_pools(max_results: 10)
    user_pools = []
    paginator.each_page do |page|
      user_pools.concat(page.user_pools)
    end

    if user_pools.empty?
      @logger.info('No Cognito user pools found.')
    else
      user_pools.each do |user_pool|
        @logger.info("User pool ID: #{user_pool.id}")
        @logger.info("User pool name: #{user_pool.name}")
        @logger.info("User pool status: #{user_pool.status}")
        @logger.info('---')
      end
    end
  end
end
```

```
if $PROGRAM_NAME == __FILE__
    cognito_client = Aws::CognitoIdentityProvider::Client.new
    manager = CognitoManager.new(cognito_client)
    manager.list_user_pools
end
```

- For API details, see [ListUserPools](#) in *AWS SDK for Ruby API Reference*.

Code examples

- [Basic examples for Amazon Cognito Identity Provider using AWS SDKs](#)

- [Hello Amazon Cognito](#)

- [Actions for Amazon Cognito Identity Provider using AWS SDKs](#)

- [Use AdminCreateUser with an AWS SDK or CLI](#)
 - [Use Admin GetUser with an AWS SDK or CLI](#)
 - [Use AdminInitiateAuth with an AWS SDK or CLI](#)
 - [Use AdminRespondToAuthChallenge with an AWS SDK](#)
 - [Use AdminSetUserPassword with an AWS SDK](#)
 - [Use AssociateSoftwareToken with an AWS SDK](#)
 - [Use ConfirmDevice with an AWS SDK](#)
 - [Use ConfirmForgotPassword with an AWS SDK or CLI](#)
 - [Use ConfirmSignUp with an AWS SDK or CLI](#)
 - [Use CreateUserPool with an AWS SDK or CLI](#)
 - [Use CreateUserPoolClient with an AWS SDK or CLI](#)
 - [Use DeleteUser with an AWS SDK or CLI](#)
 - [Use ForgotPassword with an AWS SDK or CLI](#)
 - [Use InitiateAuth with an AWS SDK](#)
 - [Use ListUserPools with an AWS SDK or CLI](#)
 - [Use ListUsers with an AWS SDK or CLI](#)
 - [Use ResendConfirmationCode with an AWS SDK or CLI](#)

- [Use SignUp with an AWS SDK or CLI](#)
 - [Use UpdateUserPool with an AWS SDK or CLI](#)
 - [Use VerifySoftwareToken with an AWS SDK](#)
- [Scenarios for Amazon Cognito Identity Provider using AWS SDKs](#)
 - [Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK](#)
 - [Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK](#)
 - [Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK](#)
 - [Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK](#)

Basic examples for Amazon Cognito Identity Provider using AWS SDKs

The following code examples show how to use the basics of Amazon Cognito Identity Provider with AWS SDKs.

Examples

- [Hello Amazon Cognito](#)
- [Actions for Amazon Cognito Identity Provider using AWS SDKs](#)
 - [Use AdminCreateUser with an AWS SDK or CLI](#)
 - [Use Admin GetUser with an AWS SDK or CLI](#)
 - [Use AdminInitiateAuth with an AWS SDK or CLI](#)
 - [Use AdminRespondToAuthChallenge with an AWS SDK](#)
 - [Use AdminSetUserPassword with an AWS SDK](#)
 - [Use AssociateSoftwareToken with an AWS SDK](#)
 - [Use ConfirmDevice with an AWS SDK](#)
 - [Use ConfirmForgotPassword with an AWS SDK or CLI](#)
 - [Use ConfirmSignUp with an AWS SDK or CLI](#)
 - [Use CreateUserPool with an AWS SDK or CLI](#)
 - [Use CreateUserPoolClient with an AWS SDK or CLI](#)
- Basics • [Use DeleteUser with an AWS SDK or CLI](#)

- [Use ForgotPassword with an AWS SDK or CLI](#)
- [Use InitiateAuth with an AWS SDK](#)
- [Use ListUserPools with an AWS SDK or CLI](#)
- [Use ListUsers with an AWS SDK or CLI](#)
- [Use ResendConfirmationCode with an AWS SDK or CLI](#)
- [Use RespondToAuthChallenge with an AWS SDK or CLI](#)
- [Use SignUp with an AWS SDK or CLI](#)
- [Use UpdateUserPool with an AWS SDK or CLI](#)
- [Use VerifySoftwareToken with an AWS SDK](#)

Hello Amazon Cognito

The following code examples show how to get started using Amazon Cognito.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS cognito-idp)

# Set this project's name.
project("hello_cognito")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)
```

```
# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
    # libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
"${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
    # for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
    # may need to uncomment this
                                # and set the proper subdirectory to the
    # executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
    hello_cognito.cpp)

target_link_libraries(${PROJECT_NAME}
    ${AWSSDK_LINK_LIBRARIES})
```

Code for the `hello_cognito.cpp` source file.

```
#include <aws/core/Aws.h>
#include <aws/cognito-idp/CognitoIdentityProviderClient.h>
#include <aws/cognito-idp/model/ListUserPoolsRequest.h>
#include <iostream>

/*
```

```
* A "Hello Cognito" starter application which initializes an Amazon Cognito
client and lists the Amazon Cognito
user pools.
*
* main function
*
* Usage: 'hello_cognito'
*
*/
```

```
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//    options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
cognitoClient(clientConfig);

        Aws::String nextToken; // Used for pagination.
        std::vector<Aws::String> userPools;

        do {
            Aws::CognitoIdentityProvider::Model::ListUserPoolsRequest
listUserPoolsRequest;
            if (!nextToken.empty()) {
                listUserPoolsRequest.SetNextToken(nextToken);
            }

            Aws::CognitoIdentityProvider::Model::ListUserPoolsOutcome
listUserPoolsOutcome =
                cognitoClient.ListUserPools(listUserPoolsRequest);

            if (listUserPoolsOutcome.IsSuccess()) {
                for (auto &userPool:
listUserPoolsOutcome.GetResult().GetUserPools() {

                    userPools.push_back(userPool.GetName());
                }
            }
        }
    }
}
```

```
        nextToken = listUserPoolsOutcome.GetResult().GetNextToken();
    } else {
        std::cerr << "ListUserPools error: " <<
listUserPoolsOutcome.GetError().GetMessage() << std::endl;
        result = 1;
        break;
    }

} while (!nextToken.empty());
std::cout << userPools.size() << " user pools found." << std::endl;
for (auto &userPool: userPools) {
    std::cout << "    user pool: " << userPool << std::endl;
}
}

Aws::ShutdownAPI(options); // Should only be called once.
return result;
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
package main

import (
    "context"
    "fmt"
    "log"
```

```
"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
"github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
// Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)
    fmt.Println("Let's list the user pools for your account.")
    var pools []types.UserPoolDescriptionType
    paginator := cognitoidentityprovider.NewListUserPoolsPaginator(
        cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:
aws.Int32(10)})
    for paginator.HasMorePages() {
        output, err := paginator.NextPage(ctx)
        if err != nil {
            log.Printf("Couldn't get user pools. Here's why: %v\n", err)
        } else {
            pools = append(pools, output.UserPools...)
        }
    }
    if len(pools) == 0 {
        fmt.Println("You don't have any user pools!")
    } else {
        for _, pool := range pools {
            fmt.Printf("\t%v: %v\n", *pool.Name, *pool.Id)
        }
    }
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListUserPools {
    public static void main(String[] args) {
        CognitoIdentityProviderClient cognitoClient =
CognitoIdentityProviderClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllUserPools(cognitoClient);
        cognitoClient.close();
    }
}
```

```
}

    public static void listAllUserPools(CognitoIdentityProviderClient
cognitoClient) {
    try {
        ListUserPoolsRequest request = ListUserPoolsRequest.builder()
            .maxResults(10)
            .build();

        ListUserPoolsResponse response =
cognitoClient.listUserPools(request);
        response.userPools().forEach(userpool -> {
            System.out.println("User pool " + userpool.name() + ", User ID "
+ userpool.id());
        });
    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});
```

```
export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- For API details, see [ListUserPools](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import boto3

# Create a Cognito Identity Provider client
cognitoidp = boto3.client("cognito-identity-provider")

# Initialize a paginator for the list_user_pools operation
paginator = cognitoidp.get_paginator("list_user_pools")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(MaxResults=10)

# Initialize variables for pagination
```

```
user_pools = []

# Handle pagination
for page in page_iterator:
    user_pools.extend(page.get("UserPools", []))

# Print the list of user pools
print("User Pools for the account:")
if user_pools:
    for pool in user_pools:
        print(f"Name: {pool['Name']}, ID: {pool['Id']}")
else:
    print("No user pools found.")
```

- For API details, see [ListUserPools](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require 'aws-sdk-cognitoidentityprovider'
require 'logger'

# CognitoManager is a class responsible for managing AWS Cognito operations
# such as listing all user pools in the current AWS account.
class CognitoManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all user pools associated with the AWS account.
  def list_user_pools
```

```
paginator = @client.list_user_pools(max_results: 10)
user_pools = []
paginator.each_page do |page|
  user_pools.concat(page.user_pools)
end

if user_pools.empty?
  @logger.info('No Cognito user pools found.')
else
  user_pools.each do |user_pool|
    @logger.info("User pool ID: #{user_pool.id}")
    @logger.info("User pool name: #{user_pool.name}")
    @logger.info("User pool status: #{user_pool.status}")
    @logger.info('---')
  end
end
end

if $PROGRAM_NAME == __FILE__
  cognito_client = Aws::CognitoIdentityProvider::Client.new
  manager = CognitoManager.new(cognito_client)
  manager.list_user_pools
end
```

- For API details, see [ListUserPools](#) in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Actions for Amazon Cognito Identity Provider using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Cognito Identity Provider actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon Cognito Identity Provider API and are code excerpts from larger programs that must be run in context. You can see actions in context in [Scenarios for Amazon Cognito Identity Provider using AWS SDKs](#).

The following examples include only the most commonly used actions. For a complete list, see the [Amazon Cognito Identity Provider API Reference](#).

Examples

- [Use AdminCreateUser with an AWS SDK or CLI](#)
- [Use Admin GetUser with an AWS SDK or CLI](#)
- [Use AdminInitiateAuth with an AWS SDK or CLI](#)
- [Use AdminRespondToAuthChallenge with an AWS SDK](#)
- [Use AdminSetUserPassword with an AWS SDK](#)
- [Use AssociateSoftwareToken with an AWS SDK](#)
- [Use ConfirmDevice with an AWS SDK](#)
- [Use ConfirmForgotPassword with an AWS SDK or CLI](#)
- [Use ConfirmSignUp with an AWS SDK or CLI](#)
- [Use CreateUserPool with an AWS SDK or CLI](#)
- [Use CreateUserPoolClient with an AWS SDK or CLI](#)
- [Use DeleteUser with an AWS SDK or CLI](#)
- [Use ForgotPassword with an AWS SDK or CLI](#)
- [Use InitiateAuth with an AWS SDK](#)
- [Use ListUserPools with an AWS SDK or CLI](#)
- [Use ListUsers with an AWS SDK or CLI](#)
- [Use ResendConfirmationCode with an AWS SDK or CLI](#)
- [Use RespondToAuthChallenge with an AWS SDK or CLI](#)
- [Use SignUp with an AWS SDK or CLI](#)
- [Use UpdateUserPool with an AWS SDK or CLI](#)
- [Use VerifySoftwareToken with an AWS SDK](#)

Use AdminCreateUser with an AWS SDK or CLI

The following code examples show how to use AdminCreateUser.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Write custom activity data with a Lambda function after Amazon Cognito user authentication](#)

CLI

AWS CLI

To create a user

The following `admin-create-user` example creates a user with the specified settings email address and phone number.

```
aws cognito-identity admin-create-user \
    --user-pool-id us-west-2_aaaaaaaaaa \
    --username diego \
    --user-attributes Name=email,Value=diego@example.com
    Name=phone_number,Value="+15555551212" \
    --message-action SUPPRESS
```

Output:

```
{  
  "User": {  
    "Username": "diego",  
    "Attributes": [  
      {  
        "Name": "sub",  
        "Value": "7325c1de-b05b-4f84-b321-9adc6e61f4a2"  
      },  
      {  
        "Name": "phone_number",  
        "Value": "+15555551212"  
      },  
      {  
        "Name": "email",  
        "Value": "diego@example.com"  
      }  
    ],  
    "UserCreateDate": 1548099495.428,  
    "UserLastModifiedDate": 1548099495.428,  
    "Enabled": true,  
    "UserStatus": "FORCE_CHANGE_PASSWORD"  
  }  
}
```

{

- For API details, see [AdminCreateUser](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
    string, userName string, userEmail string) error {
    _, err := actor.CognitoClient.AdminCreateUser(ctx,
        &cognitoidentityprovider.AdminCreateUserInput{
            UserPoolId:      aws.String(userPoolId),
            Username:       aws.String(userName),
            MessageAction:  types.MessageActionTypeSuppress,
```

```
UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:  
aws.String(userEmail)}},  
}  
if err != nil {  
    var userExists *types.UsernameExistsException  
    if errors.As(err, &userExists) {  
        log.Printf("User %v already exists in the user pool.", userName)  
        err = nil  
    } else {  
        log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)  
    }  
}  
return err  
}
```

- For API details, see [AdminCreateUser](#) in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use Admin GetUser with an AWS SDK or CLI

The following code examples show how to use Admin GetUser.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get the specified user from an Amazon Cognito user pool with
/// administrator access.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
/// <returns>Async task.</returns>
public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
{
    Admin GetUserRequest userRequest = new Admin GetUserRequest
    {
        Username = userName,
        UserPoolId = poolId,
    };

    var response = await _cognitoService.Admin GetUserAsync(userRequest);

    Console.WriteLine($"User status {response.UserStatus}");
    return response.UserStatus;
}
```

- For API details, see [Admin GetUser](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";
```

```
Aws::CognitoIdentityProvider::CognitoIdentityProviderClient  
client(clientConfig);  
  
Aws::CognitoIdentityProvider::Model::Admin GetUserRequest request;  
request.SetUsername(userName);  
request.SetUserPoolId(userPoolID);  
  
Aws::CognitoIdentityProvider::Model::Admin GetUserOutcome outcome =  
    client.Admin GetUser(request);  
  
if (outcome.IsSuccess()) {  
    std::cout << "The status for " << userName << " is " <<  
  
    Aws::CognitoIdentityProvider::Model::UserStatusTypeMapper::GetNameForUserStatusType(  
        outcome.GetResult().GetUserStatus()) << std::endl;  
    std::cout << "Enabled is " << outcome.GetResult().GetEnabled() <<  
    std::endl;  
}  
else {  
    std::cerr << "Error with CognitoIdentityProvider::Admin GetUser. "  
        << outcome.GetError().GetMessage()  
        << std::endl;  
}
```

- For API details, see [Admin GetUser](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To get a user

This example gets information about username jane@example.com.

Command:

```
aws cognito-idp admin-get-user --user-pool-id us-west-2_aaaaaaaaaa --  
username jane@example.com
```

Output:

```
{
```

```
"Username": "4320de44-2322-4620-999b-5e2e1c8df013",
"Enabled": true,
"UserStatus": "FORCE_CHANGE_PASSWORD",
"UserCreateDate": 1548108509.537,
"UserAttributes": [
  {
    "Name": "sub",
    "Value": "4320de44-2322-4620-999b-5e2e1c8df013"
  },
  {
    "Name": "email_verified",
    "Value": "true"
  },
  {
    "Name": "phone_number_verified",
    "Value": "true"
  },
  {
    "Name": "phone_number",
    "Value": "+01115551212"
  },
  {
    "Name": "email",
    "Value": "jane@example.com"
  }
],
"UserLastModifiedDate": 1548108509.537
}
```

- For API details, see [Admin GetUser](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void getAdminUser(CognitoIdentityProviderClient
identityProviderClient, String userName,
    String poolId) {
    try {
        Admin GetUserRequest userRequest = Admin GetUserRequest.builder()
            .username(userName)
            .userPoolId(poolId)
            .build();

        Admin GetUserResponse response =
identityProviderClient.admin GetUser(userRequest);
        System.out.println("User status " + response.userStatusAsString());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [Admin GetUser](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const admin GetUser = ({ userPoolId, username }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new Admin GetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
    });

    return client.send(command);
```

```
};
```

- For API details, see [Admin GetUser](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun getAdminUser(
    userNameVal: String?,
    poolIdVal: String?,
) {
    val userRequest =
        Admin GetUserRequest {
            username = userNameVal
            userPoolId = poolIdVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val response = identityProviderClient.admin GetUser(userRequest)
        println("User status ${response.userStatus}")
    }
}
```

- For API details, see [Admin GetUser](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
    def sign_up_user(self, user_name, password, user_email):  
        """  
        Signs up a new user with Amazon Cognito. This action prompts Amazon  
        Cognito  
        to send an email to the specified email address. The email contains a  
        code that  
        can be used to confirm the user.  
  
        When the user already exists, the user status is checked to determine  
        whether  
        the user has been confirmed.  
  
        :param user_name: The user name that identifies the new user.  
        """
```

```
:param password: The password for the new user.
:param user_email: The email address for the new user.
:return: True when the user is already confirmed with Amazon Cognito.
        Otherwise, false.
"""
try:
    kwargs = {
        "ClientId": self.client_id,
        "Username": user_name,
        "Password": password,
        "UserAttributes": [{"Name": "email", "Value": user_email}],
    }
    if self.client_secret is not None:
        kwargs["SecretHash"] = self._secret_hash(user_name)
    response = self.cognito_idp_client.sign_up(**kwargs)
    confirmed = response["UserConfirmed"]
except ClientError as err:
    if err.response["Error"]["Code"] == "UsernameExistsException":
        response = self.cognito_idp_client.admin_get_user(
            UserPoolId=self.user_pool_id, Username=user_name
        )
        logger.warning(
            "User %s exists and is %s.", user_name,
            response["UserStatus"]
        )
        confirmed = response["UserStatus"] == "CONFIRMED"
    else:
        logger.error(
            "Couldn't sign up %s. Here's why: %s: %s",
            user_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
return confirmed
```

- For API details, see [Admin GetUser](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use AdminInitiateAuth with an AWS SDK or CLI

The following code examples show how to use AdminInitiateAuth.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Initiate an admin auth request.
/// </summary>
/// <param name="clientId">The client ID to use.</param>
/// <param name="userPoolId">The ID of the user pool.</param>
/// <param name="userName">The username to authenticate.</param>
/// <param name="password">The user's password.</param>
/// <returns>The session to use in challenge-response.</returns>
public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var request = new AdminInitiateAuthRequest
    {
        ClientId = clientId,
```

```
UserPoolId = userPoolId,  
AuthParameters = authParameters,  
AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,  
};  
  
var response = await _cognitoService.AdminInitiateAuthAsync(request);  
return response.Session;  
}
```

- For API details, see [AdminInitiateAuth](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;  
// Optional: Set to the AWS Region (overrides config file).  
// clientConfig.region = "us-east-1";  
  
Aws::CognitoIdentityProvider::CognitoIdentityProviderClient  
client(clientConfig);  
  
Aws::CognitoIdentityProvider::Model::AdminInitiateAuthRequest request;  
request.SetClientId(clientID);  
request.SetUserPoolId(userPoolID);  
request.AddAuthParameters("USERNAME", userName);  
request.AddAuthParameters("PASSWORD", password);  
request.SetAuthFlow(  
  
Aws::CognitoIdentityProvider::Model::AuthFlowType::ADMIN_USER_PASSWORD_AUTH);  
  
Aws::CognitoIdentityProvider::Model::AdminInitiateAuthOutcome outcome =  
client.AdminInitiateAuth(request);
```

```
if (outcome.IsSuccess()) {
    std::cout << "Call to AdminInitiateAuth was successful." << std::endl;
    sessionResult = outcome.GetResult().GetSession();
}
else {
    std::cerr << "Error with CognitoIdentityProvider::AdminInitiateAuth. "
        << outcome.GetError().GetMessage()
        << std::endl;
}
```

- For API details, see [AdminInitiateAuth](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To initiate authorization

This example initiates authorization using the ADMIN_NO_SRP_AUTH flow for username `jane@example.com`

The client must have sign-in API for server-based authentication (ADMIN_NO_SRP_AUTH) enabled.

Use the session information in the return value to call admin-respond-to-auth-challenge.

Command:

```
aws cognito-idp admin-initiate-auth --user-pool-id us-west-2_aaaaaaaaaa --
client-id 3n4b5urk1ft4fl3mg5e62d9ado --auth-flow ADMIN_NO_SRP_AUTH --auth-
parameters USERNAME=jane@example.com,PASSWORD=password
```

Output:

```
{
    "ChallengeName": "NEW_PASSWORD_REQUIRED",
    "Session": "SESSION",
    "ChallengeParameters": {
        "USER_ID_FOR_SRP": "84514837-dcbc-4af1-abff-f3c109334894",
        "requiredAttributes": "[]",
```

```
        "userAttributes": "{\"email_verified\": \"true\", \"phone_number_verified\": \"true\", \"phone_number\": \"+01xxx5550100\", \"email\": \"jane@example.com\"}"  
    }  
}
```

- For API details, see [AdminInitiateAuth](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static AdminInitiateAuthResponse  
initiateAuth(CognitoIdentityProviderClient identityProviderClient,  
             String clientId, String userName, String password, String userPoolId)  
{  
    try {  
        Map<String, String> authParameters = new HashMap<>();  
        authParameters.put("USERNAME", userName);  
        authParameters.put("PASSWORD", password);  
  
        AdminInitiateAuthRequest authRequest =  
AdminInitiateAuthRequest.builder()  
            .clientId(clientId)  
            .userPoolId(userPoolId)  
            .authParameters(authParameters)  
            .authFlow(AuthFlowType.ADMIN_USER_PASSWORD_AUTH)  
            .build();  
  
        AdminInitiateAuthResponse response =  
identityProviderClient.adminInitiateAuth(authRequest);  
        System.out.println("Result Challenge is : " +  
response.challengeName());  
        return response;  
  
    } catch (CognitoIdentityProviderException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
    }  
}
```

```
        System.exit(1);
    }

    return null;
}
```

- For API details, see [AdminInitiateAuth](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};
```

- For API details, see [AdminInitiateAuth](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun checkAuthMethod(
    clientIdVal: String,
    userNameVal: String,
    passwordVal: String,
    userPoolIdVal: String,
): AdminInitiateAuthResponse {
    val authParas = mutableMapOf<String, String>()
    authParas["USERNAME"] = userNameVal
    authParas["PASSWORD"] = passwordVal

    val authRequest =
        AdminInitiateAuthRequest {
            clientId = clientIdVal
            userPoolId = userPoolIdVal
            authParameters = authParas
            authFlow = AuthFlowType.AdminUserPasswordAuth
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val response = identityProviderClient.adminInitiateAuth(authRequest)
        println("Result Challenge is ${response.challengeName}")
        return response
    }
}
```

- For API details, see [AdminInitiateAuth](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
    def start_sign_in(self, user_name, password):  
        """  
        Starts the sign-in process for a user by using administrator credentials.  
        This method of signing in is appropriate for code running on a secure  
        server.  
          
        If the user pool is configured to require MFA and this is the first sign-  
        in  
        for the user, Amazon Cognito returns a challenge response to set up an  
        MFA application. When this occurs, this function gets an MFA secret from  
        Amazon Cognito and returns it to the caller.  
          
        :param user_name: The name of the user to sign in.  
        """
```

```
:param password: The user's password.  
:return: The result of the sign-in attempt. When sign-in is successful,  
this  
    returns an access token that can be used to get AWS credentials.  
Otherwise,  
    Amazon Cognito returns a challenge to set up an MFA application,  
    or a challenge to enter an MFA code from a registered MFA  
application.  
"""  
try:  
    kwargs = {  
        "UserPoolId": self.user_pool_id,  
        "ClientId": self.client_id,  
        "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",  
        "AuthParameters": {"USERNAME": user_name, "PASSWORD": password},  
    }  
    if self.client_secret is not None:  
        kwargs["AuthParameters"]["SECRET_HASH"] =  
self._secret_hash(user_name)  
        response = self.cognito_idp_client.admin_initiate_auth(**kwargs)  
        challenge_name = response.get("ChallengeName", None)  
        if challenge_name == "MFA_SETUP":  
            if (  
                "SOFTWARE_TOKEN_MFA"  
                in response["ChallengeParameters"]["MFAS_CAN_SETUP"]  
            ):  
                response.update(self.get_mfa_secret(response["Session"]))  
            else:  
                raise RuntimeError(  
                    "The user pool requires MFA setup, but the user pool is  
not "  
                    "configured for TOTP MFA. This example requires TOTP  
MFA."  
                )  
        except ClientError as err:  
            logger.error(  
                "Couldn't start sign in for %s. Here's why: %s: %s",  
                user_name,  
                err.response["Error"]["Code"],  
                err.response["Error"]["Message"],  
            )  
            raise  
    else:  
        response.pop("ResponseMetadata", None)
```

```
    return response
```

- For API details, see [AdminInitiateAuth](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use AdminRespondToAuthChallenge with an AWS SDK

The following code examples show how to use AdminRespondToAuthChallenge.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Respond to an admin authentication challenge.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="clientId">The client ID.</param>
/// <param name="mfaCode">The multi-factor authentication code.</param>
/// <param name="session">The current application session.</param>
/// <param name="clientId">The user pool ID.</param>
/// <returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
```

```
        string mfaCode,
        string session,
        string userPoolId)
    {
        Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

        var challengeResponses = new Dictionary<string, string>();
        challengeResponses.Add("USERNAME", userName);
        challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        var respondToAuthChallengeRequest = new
AdminRespondToAuthChallengeRequest
{
    ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ClientId = clientId,
    ChallengeResponses = challengeResponses,
    Session = session,
    UserPoolId = userPoolId,
};

        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
        return response.AuthenticationResult;
    }
}
```

- For API details, see [AdminRespondToAuthChallenge](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
```

```
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeRequest
request;
request.AddChallengeResponses("USERNAME", userName);
request.AddChallengeResponses("SOFTWARE_TOKEN_MFA_CODE", mfaCode);
request.SetChallengeName(
    Aws::CognitoIdentityProvider::Model::ChallengeNameType::SOFTWARE_TOKEN_MFA);
request.SetClientId(clientID);
request.SetUserPoolId(userPoolID);
request.SetSession(session);

Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeOutcome
outcome =
    client.AdminRespondToAuthChallenge(request);

if (outcome.IsSuccess()) {
    std::cout << "Here is the response to the challenge.\n" <<
    outcome.GetResult().GetAuthenticationResult().Jsonize().View().WriteReadable()
        << std::endl;

    accessToken =
        outcome.GetResult().GetAuthenticationResult().GetAccessToken();
}
else {
    std::cerr << "Error with
CognitoIdentityProvider::AdminRespondToAuthChallenge. "
        << outcome.GetError().GetMessage()
        << std::endl;
    return false;
}
```

- For API details, see [AdminRespondToAuthChallenge](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Respond to an authentication challenge.
public static void adminRespondToAuthChallenge(CognitoIdentityProviderClient
identityProviderClient,
    String userName, String clientId, String mfaCode, String session) {
    System.out.println("SOFTWARE_TOKEN_MFA challenge is generated");
    Map<String, String> challengeResponses = new HashMap<>();

    challengeResponses.put("USERNAME", userName);
    challengeResponses.put("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

    AdminRespondToAuthChallengeRequest respondToAuthChallengeRequest =
AdminRespondToAuthChallengeRequest.builder()
        .challengeName(ChallengeNameType.SOFTWARE_TOKEN_MFA)
        .clientId(clientId)
        .challengeResponses(challengeResponses)
        .session(session)
        .build();

    AdminRespondToAuthChallengeResponse respondToAuthChallengeResult =
identityProviderClient
        .adminRespondToAuthChallenge(respondToAuthChallengeRequest);

    System.out.println("respondToAuthChallengeResult.getAuthenticationResult()"
        + respondToAuthChallengeResult.authenticationResult());
}
```

- For API details, see [AdminRespondToAuthChallenge](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const adminRespondToAuthChallenge = ({  
    userPoolId,  
    clientId,  
    username,  
    totp,  
    session,  
) => {  
    const client = new CognitoIdentityProviderClient({});  
    const command = new AdminRespondToAuthChallengeCommand({  
        ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,  
        ChallengeResponses: {  
            SOFTWARE_TOKEN_MFA_CODE: totp,  
            USERNAME: username,  
        },  
        ClientId: clientId,  
        UserPoolId: userPoolId,  
        Session: session,  
    });  
  
    return client.send(command);  
};
```

- For API details, see [AdminRespondToAuthChallenge](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Respond to an authentication challenge.
suspend fun adminRespondToAuthChallenge(
    userName: String,
    clientIdVal: String?,
    mfaCode: String,
    sessionVal: String?,
) {
    println("SOFTWARE_TOKEN_MFA challenge is generated")
    val challengeResponses0b = mutableMapOf<String, String>()
    challengeResponses0b["USERNAME"] = userName
    challengeResponses0b["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

    val adminRespondToAuthChallengeRequest =
        AdminRespondToAuthChallengeRequest {
            challengeName = ChallengeNameType.SoftwareTokenMfa
            clientId = clientIdVal
            challengeResponses = challengeResponses0b
            session = sessionVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val respondToAuthChallengeResult =
            identityProviderClient.adminRespondToAuthChallenge(adminRespondToAuthChallengeRequest)
        println("respondToAuthChallengeResult.getAuthenticationResult()\n${respondToAuthChallengeResult.authenticationResult}")
    }
}
```

- For API details, see [AdminRespondToAuthChallenge](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Respond to an MFA challenge by providing a code generated by an associated MFA application.

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
    def respond_to_mfa_challenge(self, user_name, session, mfa_code):  
        """  
        Responds to a challenge for an MFA code. This completes the second step  
        of  
        a two-factor sign-in. When sign-in is successful, it returns an access  
        token  
        that can be used to get AWS credentials from Amazon Cognito.  
  
        :param user_name: The name of the user who is signing in.  
        """
```

```
:param session: Session information returned from a previous call to
initiate
    authentication.
:param mfa_code: A code generated by the associated MFA application.
:return: The result of the authentication. When successful, this contains
an
    access token for the user.

"""

try:
    kwargs = {
        "UserPoolId": self.user_pool_id,
        "ClientId": self.client_id,
        "ChallengeName": "SOFTWARE_TOKEN_MFA",
        "Session": session,
        "ChallengeResponses": {
            "USERNAME": user_name,
            "SOFTWARE_TOKEN_MFA_CODE": mfa_code,
        },
    }
    if self.client_secret is not None:
        kwargs["ChallengeResponses"]["SECRET_HASH"] = self._secret_hash(
            user_name
        )
    response =
self.cognito_idp_client.admin_respond_to_auth_challenge(**kwargs)
    auth_result = response["AuthenticationResult"]
except ClientError as err:
    if err.response["Error"]["Code"] == "ExpiredCodeException":
        logger.warning(
            "Your MFA code has expired or has been used already. You
might have "
            "to wait a few seconds until your app shows you a new code."
        )
    else:
        logger.error(
            "Couldn't respond to mfa challenge for %s. Here's why: %s:
%s",
            user_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return auth_result
```

- For API details, see [AdminRespondToAuthChallenge](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use AdminSetUserPassword with an AWS SDK

The following code example shows how to use AdminSetUserPassword.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Write custom activity data with a Lambda function after Amazon Cognito user authentication](#)

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)
```

```
type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
    string, userName string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
    &cognitoidentityprovider.AdminSetUserPasswordInput{
        Password: aws.String(password),
        UserPoolId: aws.String(userPoolId),
        Username: aws.String(userName),
        Permanent: true,
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
        }
    }
    return err
}
```

- For API details, see [AdminSetUserPassword](#) in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use AssociateSoftwareToken with an AWS SDK

The following code examples show how to use `AssociateSoftwareToken`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get an MFA token to authenticate the user with the authenticator.
/// </summary>
/// <param name="session">The session name.</param>
/// <returns>The session name.</returns>
public async Task<string> AssociateSoftwareTokenAsync(string session)
{
    var softwareTokenRequest = new AssociateSoftwareTokenRequest
    {
        Session = session,
    };

    var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
    var secretCode = tokenResponse.SecretCode;

    Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

    return tokenResponse.Session;
}
```

- For API details, see [AssociateSoftwareToken](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenRequest
request;
request.SetSession(session);

Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenOutcome
outcome =
    client.AssociateSoftwareToken(request);

if (outcome.IsSuccess()) {
    std::cout
        << "Enter this setup key into an authenticator app, for
example Google Authenticator."
        << std::endl;
    std::cout << "Setup key: " << outcome.GetResult().GetSecretCode()
        << std::endl;
#endif USING_QR
    printAsterisksLine();
    std::cout << "\nOr scan the QR code in the file '" << QR_CODE_PATH <<
    "."
        << std::endl;

    saveQRCode(std::string("otpauth://totp/") + userName + "?secret=" +
               outcome.GetResult().GetSecretCode());
#endif // USING_QR
    session = outcome.GetResult().GetSession();
```

```
    }
    else {
        std::cerr << "Error with
CognitoIdentityProvider::AssociateSoftwareToken. "
        << outcome.GetError().GetMessage()
        << std::endl;
    return false;
}
```

- For API details, see [AssociateSoftwareToken](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static String getSecretForAppMFA(CognitoIdentityProviderClient
identityProviderClient, String session) {
    AssociateSoftwareTokenRequest softwareTokenRequest =
AssociateSoftwareTokenRequest.builder()
    .session(session)
    .build();

    AssociateSoftwareTokenResponse tokenResponse = identityProviderClient
        .associateSoftwareToken(softwareTokenRequest);
    String secretCode = tokenResponse.secretCode();
    System.out.println("Enter this token into Google Authenticator");
    System.out.println(secretCode);
    return tokenResponse.session();
}
```

- For API details, see [AssociateSoftwareToken](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- For API details, see [AssociateSoftwareToken](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun getSecretForAppMFA(sessionVal: String?): String? {
  val softwareTokenRequest =
    AssociateSoftwareTokenRequest {
      session = sessionVal
    }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
  { identityProviderClient ->
```

```
    val tokenResponse =
identityProviderClient.associateSoftwareToken(softwareTokenRequest)
    val secretCode = tokenResponse.secretCode
    println("Enter this token into Google Authenticator")
    println(secretCode)
    return tokenResponse.session
}
}
```

- For API details, see [AssociateSoftwareToken](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
                 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
        client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
        pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret

    def get_mfa_secret(self, session):
```

```
"""
    Gets a token that can be used to associate an MFA application with the
    user.

    :param session: Session information returned from a previous call to
        initiate
        authentication.
    :return: An MFA token that can be used to set up an MFA application.
"""

try:
    response =
self.cognito_idp_client.associate_software_token(Session=session)
except ClientError as err:
    logger.error(
        "Couldn't get MFA secret. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    response.pop("ResponseMetadata", None)
    return response
```

- For API details, see [AssociateSoftwareToken](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ConfirmDevice with an AWS SDK

The following code examples show how to use ConfirmDevice.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Initiates and confirms tracking of the device.
/// </summary>
/// <param name="accessToken">The user's access token.</param>
/// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
/// <param name="deviceName">The device name.</param>
/// <returns></returns>
public async Task<bool> ConfirmDeviceAsync(string accessToken, string
deviceKey, string deviceName)
{
    var request = new ConfirmDeviceRequest
    {
        AccessToken = accessToken,
        DeviceKey = deviceKey,
        DeviceName = deviceName
    };

    var response = await _cognitoService.ConfirmDeviceAsync(request);
    return response.UserConfirmationNecessary;
}
```

- For API details, see [ConfirmDevice](#) in *AWS SDK for .NET API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });

  return client.send(command);
};
```

- For API details, see [ConfirmDevice](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""
```

```
def __init__(self, cognito_idp_client, user_pool_id, client_id,
client_secret=None):
    """
    :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
    client.
    :param user_pool_id: The ID of an existing Amazon Cognito user pool.
    :param client_id: The ID of a client application registered with the user
    pool.
    :param client_secret: The client secret, if the client has a secret.
    """
    self.cognito_idp_client = cognito_idp_client
    self.user_pool_id = user_pool_id
    self.client_id = client_id
    self.client_secret = client_secret

def confirm_mfa_device(
    self,
    user_name,
    device_key,
    device_group_key,
    device_password,
    access_token,
    aws_srp,
):
    """
    Confirms an MFA device to be tracked by Amazon Cognito. When a device is
    tracked, its key and password can be used to sign in without requiring a
    new
    MFA code from the MFA application.

    :param user_name: The user that is associated with the device.
    :param device_key: The key of the device, returned by Amazon Cognito.
    :param device_group_key: The group key of the device, returned by Amazon
    Cognito.
    :param device_password: The password that is associated with the device.
    :param access_token: The user's access token.
    :param aws_srp: A class that helps with Secure Remote Password (SRP)
        calculations. The scenario associated with this example
    uses
        the warrant package.
    :return: True when the user must confirm the device. Otherwise, False.
When
```

```
        False, the device is automatically confirmed and tracked.  
    """  
  
    srp_helper = aws_srp.AWSSRP(  
        username=user_name,  
        password=device_password,  
        pool_id="_",  
        client_id=self.client_id,  
        client_secret=None,  
        client=self.cognito_idp_client,  
    )  
    device_and_pw = f"{device_group_key}{device_key}:{device_password}"  
    device_and_pw_hash = aws_srp.hash_sha256(device_and_pw.encode("utf-8"))  
    salt = aws_srp.pad_hex(aws_srp.get_random(16))  
    x_value = aws_srp.hex_to_long(aws_srp.hex_hash(salt +  
device_and_pw_hash))  
    verifier = aws_srp.pad_hex(pow(srp_helper.val_g, x_value,  
srp_helper.big_n))  
    device_secret_verifier_config = {  
        "PasswordVerifier": base64.standard_b64encode(  
            bytearray.fromhex(verifier)  
        ).decode("utf-8"),  
        "Salt":  
            base64.standard_b64encode(bytearray.fromhex(salt)).decode("utf-8"),  
    }  
    try:  
        response = self.cognito_idp_client.confirm_device(  
            AccessToken=access_token,  
            DeviceKey=device_key,  
            DeviceSecretVerifierConfig=device_secret_verifier_config,  
        )  
        user_confirm = response["UserConfirmationNecessary"]  
    except ClientError as err:  
        logger.error(  
            "Couldn't confirm mfa device %s. Here's why: %s: %s",  
            device_key,  
            err.response["Error"]["Code"],  
            err.response["Error"]["Message"],  
        )  
        raise  
    else:  
        return user_confirm
```

- For API details, see [ConfirmDevice](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ConfirmForgotPassword with an AWS SDK or CLI

The following code examples show how to use ConfirmForgotPassword.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Automatically migrate known users with a Lambda function](#)

CLI

AWS CLI

To confirm a forgotten password

This example confirms a forgotten password for username diego@example.com.

Command:

```
aws cognito-identity confirm-forgot-password --client-id 3n4b5urk1ft4fl3mg5e62d9ado --username=diego@example.com --password PASSWORD --confirmation-code CONF_CODE
```

- For API details, see [ConfirmForgotPassword](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
    &cognitoidentityprovider.ConfirmForgotPasswordInput{
        ClientId:           aws.String(clientId),
        ConfirmationCode: aws.String(code),
        Password:          aws.String(password),
        Username:          aws.String(userName),
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
        }
    }
    return err
}
```

- For API details, see [ConfirmForgotPassword](#) in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ConfirmSignUp with an AWS SDK or CLI

The following code examples show how to use ConfirmSignUp.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Confirm that the user has signed up.
/// </summary>
/// <param name="clientId">The Id of this application.</param>
/// <param name="code">The confirmation code sent to the user.</param>
/// <param name="userName">The username.</param>
/// <returns>True if successful.</returns>
public async Task<bool> ConfirmSignupAsync(string clientId, string code,
string userName)
{
    var signUpRequest = new ConfirmSignUpRequest
    {
        ClientId = clientId,
        ConfirmationCode = code,
        Username = userName,
    };

    var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
```

```
if (response.HttpStatusCode == HttpStatusCode.OK)
{
    Console.WriteLine($"'{userName}' was confirmed");
    return true;
}
return false;
}
```

- For API details, see [ConfirmSignUp](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

Aws::CognitoIdentityProvider::Model::ConfirmSignUpRequest request;
request.SetClientId(clientID);
request.SetConfirmationCode(confirmationCode);
request.SetUsername(userName);

Aws::CognitoIdentityProvider::Model::ConfirmSignUpOutcome outcome =
    client.ConfirmSignUp(request);

if (outcome.IsSuccess()) {
    std::cout << "ConfirmSignup was Successful."
        << std::endl;
}
else {
```

```
        std::cerr << "Error with CognitoIdentityProvider::ConfirmSignUp. "
        << outcome.GetError().GetMessage()
        << std::endl;
    return false;
}
```

- For API details, see [ConfirmSignUp](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To confirm sign-up

This example confirms sign-up for username diego@example.com.

Command:

```
aws cognito-idp confirm-sign-up --client-id 3n4b5urk1ft4fl3mg5e62d9ado --
username=diego@example.com --confirmation-code CONF_CODE
```

- For API details, see [ConfirmSignUp](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void confirmSignUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String code,
String userName) {
try {
    ConfirmSignUpRequest signUpRequest = ConfirmSignUpRequest.builder()
```

```
.clientId(clientId)
.confirmationCode(code)
.username(userName)
.build();

identityProviderClient.confirmSignUp(signUpRequest);
System.out.println(userName + " was confirmed");

} catch (CognitoIdentityProviderException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- For API details, see [ConfirmSignUp](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const confirmSignUp = ({ clientId, username, code }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new ConfirmSignUpCommand({
        ClientId: clientId,
        Username: username,
        ConfirmationCode: code,
    });

    return client.send(command);
};
```

- For API details, see [ConfirmSignUp](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun confirmSignUp(  
    clientIdVal: String?,  
    codeVal: String?,  
    userNameVal: String?,  
) {  
    val signUpRequest =  
        ConfirmSignUpRequest {  
            clientId = clientIdVal  
            confirmationCode = codeVal  
            username = userNameVal  
        }  
  
    CognitoIdentityProviderClient { region = "us-east-1" }.use  
{ identityProviderClient ->  
    identityProviderClient.confirmSignUp(signUpRequest)  
    println("$userNameVal was confirmed")  
}  
}
```

- For API details, see [ConfirmSignUp](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
  
    def confirm_user_sign_up(self, user_name, confirmation_code):  
        """  
        Confirms a previously created user. A user must be confirmed before they  
        can sign in to Amazon Cognito.  
  
        :param user_name: The name of the user to confirm.  
        :param confirmation_code: The confirmation code sent to the user's  
        registered  
                           email address.  
        :return: True when the confirmation succeeds.  
        """  
        try:  
            kwargs = {  
                "ClientId": self.client_id,  
                "Username": user_name,  
                "ConfirmationCode": confirmation_code,  
            }  
            if self.client_secret is not None:  
                kwargs["SecretHash"] = self._secret_hash(user_name)  
            self.cognito_idp_client.confirm_sign_up(**kwargs)  
        except ClientError as err:  
            logger.error(  
                "Couldn't confirm sign up for %s. Here's why: %s: %s",  
                user_name,
```

```
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return True
```

- For API details, see [ConfirmSignUp](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `CreateUserPool` with an AWS SDK or CLI

The following code examples show how to use `CreateUserPool`.

CLI

AWS CLI

To create a minimally configured user pool

This example creates a user pool named `MyUserPool` using default values. There are no required attributes and no application clients. MFA and advanced security is disabled.

Command:

```
aws cognito-identity create-user-pool --pool-name MyUserPool
```

Output:

```
{
  "UserPool": {
    "SchemaAttributes": [
      {
        "Name": "sub",
        "StringAttributeConstraints": {
          "MinLength": "1",
          "MaxLength": "2048"
        }
      }
    ]
  }
}
```

```
        },
        "DeveloperOnlyAttribute": false,
        "Required": true,
        "Attribute DataType": "String",
        "Mutable": false
    },
    {
        "Name": "name",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "given_name",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "family_name",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "middle_name",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        }
    }
}
```

```
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "nickname",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "preferred_username",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "profile",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "picture",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        }
    }
]
```

```
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "Name": "website",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "Name": "email",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "AttributeDataType": "Boolean",
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Name": "email_verified",
        "Mutable": true
    },
    {
        "Name": "gender",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    }
}
```

```
        "Mutable": true
    },
    {
        "Name": "birthdate",
        "StringAttributeConstraints": {
            "MinLength": "10",
            "MaxLength": "10"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "zoneinfo",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "locale",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    },
    {
        "Name": "phone_number",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Attribute DataType": "String",
        "Mutable": true
    }
}
```

```
        "Mutable": true
    },
    {
        "AttributeDataType": "Boolean",
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Name": "phone_number_verified",
        "Mutable": true
    },
    {
        "Name": "address",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "Name": "updated_at",
        "NumberAttributeConstraints": {
            "MinValue": "0"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "Number",
        "Mutable": true
    }
],
"MfaConfiguration": "OFF",
"Name": "MyUserPool",
"LastModifiedDate": 154783345.777,
"AdminCreateUserConfig": {
    "UnusedAccountValidityDays": 7,
    "AllowAdminCreateUserOnly": false
},
"EmailConfiguration": {},
"Policies": {
    "PasswordPolicy": {
        "RequireLowercase": true,
        "RequireSymbols": true,
        "RequireNumbers": true,
```

```
        "MinimumLength": 8,
        "RequireUppercase": true
    },
},
"CreationDate": 1547833345.777,
"EstimatedNumberOfUsers": 0,
"Id": "us-west-2_aaaaaaaaaa",
"LambdaConfig": {}
}
}
```

To create a user pool with two required attributes

This example creates a user pool MyUserPool. The pool is configured to accept email as a username attribute. It also sets the email source address to a validated address using Amazon Simple Email Service.

Command:

```
aws cognito-identity create-user-pool --pool-name MyUserPool --username-attributes "email" --email-configuration=SourceArn="arn:aws:ses:us-east-1:111111111111:identity/jane@example.com",ReplyToEmailAddress="jane@example.com"
```

Output:

```
{
  "UserPool": {
    "SchemaAttributes": [
      {
        "Name": "sub",
        "StringAttributeConstraints": {
          "MinLength": "1",
          "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": true,
        "Attribute DataType": "String",
        "Mutable": false
      },
      {
        "Name": "name",
        "StringAttributeConstraints": {
```

```
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "given_name",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "family_name",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "middle_name",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "nickname",
    "StringAttributeConstraints": {
```

```
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "preferred_username",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "profile",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "picture",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "website",
    "StringAttributeConstraints": {
```

```
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "email",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Attribute DataType": "Boolean",
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Name": "email_verified",
    "Mutable": true
},
{
    "Name": "gender",
    "StringAttributeConstraints": {
        "MinLength": "0",
        "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "Attribute DataType": "String",
    "Mutable": true
},
{
    "Name": "birthdate",
    "StringAttributeConstraints": {
        "MinLength": "10",
        "MaxLength": "10"
    },
    "DeveloperOnlyAttribute": false,
```

```
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "Name": "zoneinfo",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "Name": "locale",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "Name": "phone_number",
        "StringAttributeConstraints": {
            "MinLength": "0",
            "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
    },
    {
        "AttributeDataType": "Boolean",
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "Name": "phone_number_verified",
        "Mutable": true
    },
}
```

```
{  
    "Name": "address",  
    "StringAttributeConstraints": {  
        "MinLength": "0",  
        "MaxLength": "2048"  
    },  
    "DeveloperOnlyAttribute": false,  
    "Required": false,  
    "Attribute DataType": "String",  
    "Mutable": true  
},  
{  
    "Name": "updated_at",  
    "NumberAttributeConstraints": {  
        "MinValue": "0"  
    },  
    "DeveloperOnlyAttribute": false,  
    "Required": false,  
    "Attribute DataType": "Number",  
    "Mutable": true  
}  
,  
"MfaConfiguration": "OFF",  
"Name": "MyUserPool",  
"LastModifiedDate": 1547837788.189,  
"AdminCreateUserConfig": {  
    "UnusedAccountValidityDays": 7,  
    "AllowAdminCreateUserOnly": false  
},  
"EmailConfiguration": {  
    "ReplyToEmailAddress": "jane@example.com",  
    "SourceArn": "arn:aws:ses:us-east-1:111111111111:identity/  
jane@example.com"  
},  
"Policies": {  
    "PasswordPolicy": {  
        "RequireLowercase": true,  
        "RequireSymbols": true,  
        "RequireNumbers": true,  
        "MinimumLength": 8,  
        "RequireUppercase": true  
    }  
},  
"UsernameAttributes": [  
]
```

```
        "email"
    ],
    "CreationDate": 1547837788.189,
    "EstimatedNumberOfUsers": 0,
    "Id": "us-west-2_aaaaaaaaaa",
    "LambdaConfig": {}
}
}
```

- For API details, see [CreateUserPool](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolRequest;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CreateUserPool {
    public static void main(String[] args) {
```

```
final String usage = """\n\n    Usage:\n        <userPoolName>\n\n    Where:\n        userPoolName - The name to give your user pool when it's\n        created.\n    """;\n\n    if (args.length != 1) {\n        System.out.println(usage);\n        System.exit(1);\n    }\n\n    String userPoolName = args[0];\n    CognitoIdentityProviderClient cognitoClient =\n    CognitoIdentityProviderClient.builder()\n        .region(Region.US_EAST_1)\n        .build();\n\n    String id = createPool(cognitoClient, userPoolName);\n    System.out.println("User pool ID: " + id);\n    cognitoClient.close();\n}\n\npublic static String createPool(CognitoIdentityProviderClient cognitoClient,\nString userPoolName) {\n    try {\n        CreateUserPoolRequest request = CreateUserPoolRequest.builder()\n            .poolName(userPoolName)\n            .build();\n\n        CreateUserPoolResponse response =\n        cognitoClient.createUserPool(request);\n        return response.userPool().id();\n\n    } catch (CognitoIdentityProviderException e) {\n        System.err.println(e.awsErrorDetails().errorMessage());\n        System.exit(1);\n    }\n    return "";\n}
```

{

- For API details, see [CreateUserPool](#) in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `CreateUserPoolClient` with an AWS SDK or CLI

The following code examples show how to use `CreateUserPoolClient`.

CLI

AWS CLI

To create a user pool client

This example creates a new user pool client with two explicit authorization flows: `USER_PASSWORD_AUTH` and `ADMIN_NO_SR_P_AUTH`.

Command:

```
aws cognito-identity create-user-pool-client --user-pool-id us-west-2_aaaaaaaaaa
  --client-name MyNewClient --no-generate-secret --explicit-auth-
  flows "USER_PASSWORD_AUTH" "ADMIN_NO_SR_P_AUTH"
```

Output:

```
{  
  "UserPoolClient": {  
    "UserPoolId": "us-west-2_aaaaaaaaaa",  
    "ClientName": "MyNewClient",  
    "ClientId": "6p3bs000no6a4ue1idruvd05ad",  
    "LastModifiedDate": 1548697449.497,  
    "CreationDate": 1548697449.497,  
    "RefreshTokenValidity": 30,  
    "ExplicitAuthFlows": [  
      "USER_PASSWORD_AUTH",  
      "ADMIN_NO_SR_P_AUTH"  
    ],  
  },  
}
```

```
        "AllowedOAuthFlowsUserPoolClient": false
    }
}
```

- For API details, see [CreateUserPoolClient](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientRequest
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientRespon

/**
 * A user pool client app is an application that authenticates with Amazon
 * Cognito user pools.
 * When you create a user pool, you can configure app clients that allow mobile
 * or web applications
 * to call API operations to authenticate users, manage user attributes and
 * profiles,
 * and implement sign-up and sign-in flows.
 *
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
```

```
public class CreateUserPoolClient {  
    public static void main(String[] args) {  
        final String usage = """  
  
            Usage:  
            <clientName> <userPoolId>\s  
  
            Where:  
            clientName - The name for the user pool client to create.  
            userPoolId - The ID for the user pool.  
        """;  
  
        if (args.length != 2) {  
            System.out.println(usage);  
            System.exit(1);  
        }  
  
        String clientName = args[0];  
        String userPoolId = args[1];  
        CognitoIdentityProviderClient cognitoClient =  
        CognitoIdentityProviderClient.builder()  
            .region(Region.US_EAST_1)  
            .build();  
  
        createPoolClient(cognitoClient, clientName, userPoolId);  
        cognitoClient.close();  
    }  
  
    public static void createPoolClient(CognitoIdentityProviderClient  
cognitoClient, String clientName,  
        String userPoolId) {  
        try {  
            CreateUserPoolClientRequest request =  
CreateUserPoolClientRequest.builder()  
                .clientName(clientName)  
                .userPoolId(userPoolId)  
                .build();  
  
            CreateUserPoolClientResponse response =  
cognitoClient.createUserPoolClient(request);  
            System.out.println("User pool " +  
response.userPoolClient().clientName() + " created. ID: "  
                + response.userPoolClient().clientId());  
        }  
    }  
}
```

```
        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see [CreateUserPoolClient](#) in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteUser` with an AWS SDK or CLI

The following code examples show how to use `DeleteUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Automatically confirm known users with a Lambda function](#)
- [Automatically migrate known users with a Lambda function](#)
- [Write custom activity data with a Lambda function after Amazon Cognito user authentication](#)

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";
```

```
Aws::CognitoIdentityProvider::CognitoIdentityProviderClient  
client(clientConfig);  
  
Aws::CognitoIdentityProvider::Model::DeleteUserRequest request;  
request.SetAccessToken(accessToken);  
  
Aws::CognitoIdentityProvider::Model::DeleteUserOutcome outcome =  
    client.DeleteUser(request);  
  
if (outcome.IsSuccess()) {  
    std::cout << "The user " << userName << " was deleted."  
        << std::endl;  
}  
else {  
    std::cerr << "Error with CognitoIdentityProvider::DeleteUser. "  
        << outcome.GetError().GetMessage()  
        << std::endl;  
}
```

- For API details, see [DeleteUser in AWS SDK for C++ API Reference](#).

CLI

AWS CLI

To delete a user

This example deletes a user.

Command:

```
aws cognito-idp delete-user --access-token ACCESS_TOKEN
```

- For API details, see [DeleteUser in AWS CLI Command Reference](#).

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
    string) error {
    _, err := actor.CognitoClient.DeleteUser(ctx,
    &cognitoidentityprovider.DeleteUserInput{
        AccessToken: aws.String(userAccessToken),
    })
    if err != nil {
        log.Printf("Couldn't delete user. Here's why: %v\n", err)
    }
    return err
}
```

- For API details, see [DeleteUser in AWS SDK for Go API Reference](#).

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/**  
 * Delete the signed-in user. Useful for allowing a user to delete their  
 * own profile.  
 * @param {{ region: string, accessToken: string }} config  
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-  
provider").DeleteUserCommandOutput | null, unknown]>}  
 */  
export const deleteUser = async ({ region, accessToken }) => {  
    try {  
        const client = new CognitoIdentityProviderClient({ region });  
        const response = await client.send(  
            new DeleteUserCommand({ AccessToken: accessToken }),  
        );  
        return [response, null];  
    } catch (err) {  
        return [null, err];  
    }  
};
```

- For API details, see [DeleteUser in AWS SDK for JavaScript API Reference](#).

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ForgotPassword` with an AWS SDK or CLI

The following code examples show how to use `ForgotPassword`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Automatically migrate known users with a Lambda function](#)

CLI

AWS CLI

To force a password change

The following `forgot-password` example sends a message to `jane@example.com` to change their password.

```
aws cognito-identity forgot-password --client-id 38fjsnc484p94kpqsnet7mpld0 --  
username jane@example.com
```

Output:

```
{  
    "CodeDeliveryDetails": {  
        "Destination": "j***@e***.com",  
        "DeliveryMedium": "EMAIL",  
        "AttributeName": "email"  
    }  
}
```

- For API details, see [ForgotPassword](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoidentityprovider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}
```

{}

- For API details, see [ForgotPassword](#) in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `InitiateAuth` with an AWS SDK

The following code examples show how to use `InitiateAuth`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Automatically confirm known users with a Lambda function](#)
- [Automatically migrate known users with a Lambda function](#)
- [Sign up a user with a user pool that requires MFA](#)
- [Write custom activity data with a Lambda function after Amazon Cognito user authentication](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Initiate authorization.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The name of the user who is authenticating.</param>
```

```
/// <param name="password">The password for the user who is authenticating.</param>
/// <returns>The response from the initiate auth request.</returns>
public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var authRequest = new InitiateAuthRequest
    {
        ClientId = clientId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.InitiateAuthAsync(authRequest);
    Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

    return response;
}
```

- For API details, see [InitiateAuth](#) in *AWS SDK for .NET API Reference*.

Go

SDK for Go V2

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"
```

```
"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
"github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// SignIn signs in a user to Amazon Cognito using a username and password
// authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
string, password string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(ctx,
    &cognitoidentityprovider.InitiateAuthInput{
        AuthFlow:      "USER_PASSWORD_AUTH",
        ClientId:     aws.String(clientId),
        AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
    })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    } else {
        authResult = output.AuthenticationResult
    }
    return authResult, err
}
```

- For API details, see [InitiateAuth in AWS SDK for Go API Reference](#).

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });

  return client.send(command);
};
```

- For API details, see [InitiateAuth](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

This example shows you how to start authentication with a tracked device. To complete sign-in, the client must respond correctly to Secure Remote Password (SRP) challenges.

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
  
    def sign_in_with_tracked_device(  
        self,  
        user_name,  
        password,  
        device_key,  
        device_group_key,  
        device_password,  
        aws_srp,  
    ):  
        """  
        Signs in to Amazon Cognito as a user who has a tracked device. Signing in  
        with a tracked device lets a user sign in without entering a new MFA  
        code.  
  
        Signing in with a tracked device requires that the client respond to the  
        SRP  
        protocol. The scenario associated with this example uses the warrant  
        package  
        to help with SRP calculations.  
  
        For more information on SRP, see https://en.wikipedia.org/wiki/Secure\_Remote\_Password\_protocol.  
  
        :param user_name: The user that is associated with the device.
```

```
:param password: The user's password.
:param device_key: The key of a tracked device.
:param device_group_key: The group key of a tracked device.
:param device_password: The password that is associated with the device.
:param aws_srp: A class that helps with SRP calculations. The scenario
    associated with this example uses the warrant package.
:return: The result of the authentication. When successful, this contains
an
    access token for the user.

"""
try:
    srp_helper = aws_srp.AWSRP(
        username=user_name,
        password=device_password,
        pool_id="_",
        client_id=self.client_id,
        client_secret=None,
        client=self.cognito_idp_client,
    )

    response_init = self.cognito_idp_client.initiate_auth(
        ClientId=self.client_id,
        AuthFlow="USER_PASSWORD_AUTH",
        AuthParameters={
            "USERNAME": user_name,
            "PASSWORD": password,
            "DEVICE_KEY": device_key,
        },
    )
    if response_init["ChallengeName"] != "DEVICE_SRP_AUTH":
        raise RuntimeError(
            f"Expected DEVICE_SRP_AUTH challenge but got
{response_init['ChallengeName']}."
    )

    auth_params = srp_helper.get_auth_params()
    auth_params["DEVICE_KEY"] = device_key
    response_auth = self.cognito_idp_client.respond_to_auth_challenge(
        ClientId=self.client_id,
        ChallengeName="DEVICE_SRP_AUTH",
        ChallengeResponses=auth_params,
    )
    if response_auth["ChallengeName"] != "DEVICE_PASSWORD_VERIFIER":
        raise RuntimeError(
```

```
f"Expected DEVICE_PASSWORD_VERIFIER challenge but got "
f"{response_init['ChallengeName']}]."
)

challenge_params = response_auth["ChallengeParameters"]
challenge_params["USER_ID_FOR_SRP"] = device_group_key + device_key
cr = srp_helper.process_challenge(challenge_params, {"USERNAME": user_name})
cr["USERNAME"] = user_name
cr["DEVICE_KEY"] = device_key
response_verifier =
self.cognito_idp_client.respond_to_auth_challenge(
    ClientId=self.client_id,
    ChallengeName="DEVICE_PASSWORD_VERIFIER",
    ChallengeResponses=cr,
)
auth_tokens = response_verifier["AuthenticationResult"]
except ClientError as err:
    logger.error(
        "Couldn't start client sign in for %s. Here's why: %s: %s",
        user_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return auth_tokens
```

- For API details, see [InitiateAuth](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ListUserPools` with an AWS SDK or CLI

The following code examples show how to use `ListUserPools`.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// List the Amazon Cognito user pools for an account.
/// </summary>
/// <returns>A list of UserPoolDescriptionType objects.</returns>
public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
{
    var userPools = new List<UserPoolDescriptionType>();

    var userPoolsPaginator = _cognitoService.Paginator.ListUserPools(new
ListUserPoolsRequest());

    await foreach (var response in userPoolsPaginator.Responses)
    {
        userPools.AddRange(response.UserPools);
    }

    return userPools;
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list user pools

This example lists up to 20 user pools.

Command:

```
aws cognito-identity list-user-pools --max-results 20
```

Output:

```
{  
    "UserPools": [  
        {  
            "CreationDate": 1547763720.822,  
            "LastModifiedDate": 1547763720.822,  
            "LambdaConfig": {},  
            "Id": "us-west-2_aaaaaaaaaaa",  
            "Name": "MyUserPool"  
        }  
    ]  
}
```

- For API details, see [ListUserPools](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
package main  
  
import (  
    "context"  
    "fmt"  
    "log"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/config"  
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"  
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
```

```
)\n\n// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification\n// Service\n// (Amazon SNS) client and list the topics in your account.\n// This example uses the default settings specified in your shared credentials\n// and config files.\nfunc main() {\n    ctx := context.Background()\n    sdkConfig, err := config.LoadDefaultConfig(ctx)\n    if err != nil {\n        fmt.Println("Couldn't load default configuration. Have you set up your AWS\naccount?")\n        fmt.Println(err)\n        return\n    }\n    cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)\n    fmt.Println("Let's list the user pools for your account.")\n    var pools []types.UserPoolDescriptionType\n    paginator := cognitoidentityprovider.NewListUserPoolsPaginator(\n        cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:\n            aws.Int32(10)})\n    for paginator.HasMorePages() {\n        output, err := paginator.NextPage(ctx)\n        if err != nil {\n            log.Printf("Couldn't get user pools. Here's why: %v\\n", err)\n        } else {\n            pools = append(pools, output.UserPools...)\n        }\n    }\n    if len(pools) == 0 {\n        fmt.Println("You don't have any user pools!")\n    } else {\n        for _, pool := range pools {\n            fmt.Printf("\\t%v: %v\\n", *pool.Name, *pool.Id)\n        }\n    }\n}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListUserPools {
    public static void main(String[] args) {
        CognitoIdentityProviderClient cognitoClient =
CognitoIdentityProviderClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllUserPools(cognitoClient);
        cognitoClient.close();
    }

    public static void listAllUserPools(CognitoIdentityProviderClient
cognitoClient) {
```

```
try {
    ListUserPoolsRequest request = ListUserPoolsRequest.builder()
        .maxResults(10)
        .build();

    ListUserPoolsResponse response =
cognitoClient.listUserPools(request);
    response.userPools().forEach(userpool -> {
        System.out.println("User pool " + userpool.name() + ", User ID "
+ userpool.id());
    });

} catch (CognitoIdentityProviderException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Java 2.x API Reference*.

Rust

SDK for Rust

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn show_pools(client: &Client) -> Result<(), Error> {
    let response = client.list_user_pools().max_results(10).send().await?;
    let pools = response.user_pools();
    println!("User pools:");
    for pool in pools {
        println!("  ID:          {}", pool.id().unwrap_or_default());
        println!("  Name:        {}", pool.name().unwrap_or_default());
        println!("  Lambda Config:  {:?}", pool.lambda_config().unwrap());
        println!(
            "  Last modified:  {}",
            pool.last_modified().unwrap(),
        );
    }
}
```

```
        pool.last_modified_date().unwrap().to_chrono_utc()?
    );
    println!(
        " Creation date: {:?}",
        pool.creation_date().unwrap().to_chrono_utc()
    );
    println!();
}
println!("Next token: {}", response.next_token().unwrap_or_default()));

Ok(())
}
```

- For API details, see [ListUserPools](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `ListUsers` with an AWS SDK or CLI

The following code examples show how to use `ListUsers`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Get a list of users for the Amazon Cognito user pool.
```

```
/// </summary>
/// <param name="userPoolId">The user pool ID.</param>
/// <returns>A list of users.</returns>
public async Task<List<UserType>> ListUsersAsync(string userPoolId)
{
    var request = new ListUsersRequest
    {
        UserPoolId = userPoolId
    };

    var users = new List<UserType>();

    var usersPaginator = _cognitoService.Paginator.ListUsers(request);
    await foreach (var response in usersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- For API details, see [ListUsers](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list users

This example lists up to 20 users.

Command:

```
aws cognito-idp list-users --user-pool-id us-west-2_aaaaaaaaaa --limit 20
```

Output:

```
{
  "Users": [
    {
      "Username": "22704aa3-fc10-479a-97eb-2af5806bd327",
```

```
        "Enabled": true,
        "UserStatus": "FORCE_CHANGE_PASSWORD",
        "UserCreateDate": 1548089817.683,
        "UserLastModifiedDate": 1548089817.683,
        "Attributes": [
            {
                "Name": "sub",
                "Value": "22704aa3-fc10-479a-97eb-2af5806bd327"
            },
            {
                "Name": "email_verified",
                "Value": "true"
            },
            {
                "Name": "email",
                "Value": "mary@example.com"
            }
        ]
    }
}
```

- For API details, see [ListUsers](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
software.amazon.awssdk.services.cognitoidentityprovider.model.ListUsersRequest;
```

```
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ListUsersResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListUsers {
    public static void main(String[] args) {

        final String usage = """
            Usage:
            <userPoolId>\s

            Where:
            userPoolId - The ID given to your user pool when it's
            created.
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String userPoolId = args[0];
        CognitoIdentityProviderClient cognitoClient =
        CognitoIdentityProviderClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllUsers(cognitoClient, userPoolId);
        listUsersFilter(cognitoClient, userPoolId);
        cognitoClient.close();
    }

    public static void listAllUsers(CognitoIdentityProviderClient cognitoClient,
        String userPoolId) {
        try {
```

```
        ListUsersRequest usersRequest = ListUsersRequest.builder()
            .userPoolId(userPoolId)
            .build();

        ListUsersResponse response = cognitoClient.listUsers(usersRequest);
        response.users().forEach(user -> {
            System.out.println("User " + user.username() + " Status " +
user.userStatus() + " Created "
                + user.userCreateDate());
        });

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

// Shows how to list users by using a filter.
public static void listUsersFilter(CognitoIdentityProviderClient
cognitoClient, String userPoolId) {

    try {
        String filter = "email = \"tblue@noserver.com\"";
        ListUsersRequest usersRequest = ListUsersRequest.builder()
            .userPoolId(userPoolId)
            .filter(filter)
            .build();

        ListUsersResponse response = cognitoClient.listUsers(usersRequest);
        response.users().forEach(user -> {
            System.out.println("User with filter applied " + user.username()
+ " Status " + user.userStatus()
                + " Created " + user.userCreateDate());
        });

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const listUsers = ({ userPoolId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- For API details, see [ListUsers](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun listAllUsers(userPoolId: String) {
  val request =
    ListUsersRequest {
      this.userPoolId = userPoolId
    }
}

CognitoIdentityProviderClient { region = "us-east-1" }.use { cognitoClient ->
```

```
    val response = cognitoClient.listUsers(request)
    response.users?.forEach { user ->
        println("The user name is ${user.username}")
    }
}
```

- For API details, see [ListUsers](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
                 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
        client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
        pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret

    def list_users(self):
        """
        Returns a list of the users in the current user pool.
        
```

```
:return: The list of users.  
"""  
try:  
    response =  
self.cognito_idp_client.list_users(UserPoolId=self.user_pool_id)  
    users = response["Users"]  
except ClientError as err:  
    logger.error(  
        "Couldn't list users for %s. Here's why: %s: %s",  
        self.user_pool_id,  
        err.response["Error"]["Code"],  
        err.response["Error"]["Message"],  
    )  
    raise  
else:  
    return users
```

- For API details, see [ListUsers](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ResendConfirmationCode with an AWS SDK or CLI

The following code examples show how to use ResendConfirmationCode.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Send a new confirmation code to a user.
/// </summary>
/// <param name="clientId">The Id of the client application.</param>
/// <param name="userName">The username of user who will receive the code.</param>
/// <returns>The delivery details.</returns>
public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string clientId, string userName)
{
    var codeRequest = new ResendConfirmationCodeRequest
    {
        ClientId = clientId,
        Username = userName,
    };

    var response = await
_cognitoService.ResendConfirmationCodeAsync(codeRequest);

    Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

    return response.CodeDeliveryDetails;
}
```

- For API details, see [ResendConfirmationCode](#) in *AWS SDK for .NET API Reference*.

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeRequest
request;
request.SetUsername(userName);
request.SetClientId(clientID);

Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeOutcome
outcome =
    client.ResendConfirmationCode(request);

if (outcome.IsSuccess()) {
    std::cout
        << "CognitoIdentityProvider::ResendConfirmationCode was
successful."
        << std::endl;
}
else {
    std::cerr << "Error with
CognitoIdentityProvider::ResendConfirmationCode. "
        << outcome.GetError().GetMessage()
        << std::endl;
    return false;
}
```

- For API details, see [ResendConfirmationCode](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To resend a confirmation code

The following `resend-confirmation-code` example sends a confirmation code to the user `jane`.

```
aws cognito-identity resend-confirmation-code \
--client-id 12a3b456c7de890f11g123hijk \
--username jane
```

Output:

```
{  
    "CodeDeliveryDetails": {  
        "Destination": "j***@e***.com",  
        "DeliveryMedium": "EMAIL",  
        "AttributeName": "email"  
    }  
}
```

For more information, see [Signing up and confirming user accounts](#) in the *Amazon Cognito Developer Guide*.

- For API details, see [ResendConfirmationCode](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void resendConfirmationCode(CognitoIdentityProviderClient  
identityProviderClient, String clientId,  
String userName) {
```

```
try {
    ResendConfirmationCodeRequest codeRequest =
ResendConfirmationCodeRequest.builder()
    .clientId(clientId)
    .username(userName)
    .build();

    ResendConfirmationCodeResponse response =
identityProviderClient.resendConfirmationCode(codeRequest);
    System.out.println("Method of delivery is " +
response.codeDeliveryDetails().deliveryMediumAsString());

} catch (CognitoIdentityProviderException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- For API details, see [ResendConfirmationCode](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const resendConfirmationCode = ({ clientId, username }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new ResendConfirmationCodeCommand({
        ClientId: clientId,
        Username: username,
    });

    return client.send(command);
};
```

- For API details, see [ResendConfirmationCode](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun resendConfirmationCode(
    clientIdVal: String?,
    userNameVal: String?,
) {
    val codeRequest =
        ResendConfirmationCodeRequest {
            clientId = clientIdVal
            username = userNameVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val response = identityProviderClient.resendConfirmationCode(codeRequest)
        println("Method of delivery is " +
        (response.codeDeliveryDetails?.deliveryMedium))
    }
}
```

- For API details, see [ResendConfirmationCode](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
    def resend_confirmation(self, user_name):  
        """  
        Prompts Amazon Cognito to resend an email with a new confirmation code.  
  
        :param user_name: The name of the user who will receive the email.  
        :return: Delivery information about where the email is sent.  
        """  
        try:  
            kwargs = {"ClientId": self.client_id, "Username": user_name}  
            if self.client_secret is not None:  
                kwargs["SecretHash"] = self._secret_hash(user_name)  
            response = self.cognito_idp_client.resend_confirmation_code(**kwargs)  
            delivery = response["CodeDeliveryDetails"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't resend confirmation to %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
    else:
        return delivery
```

- For API details, see [ResendConfirmationCode](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use RespondToAuthChallenge with an AWS SDK or CLI

The following code examples show how to use RespondToAuthChallenge.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

CLI

AWS CLI

To respond to an authorization challenge

This example responds to an authorization challenge initiated with initiate-auth. It is a response to the NEW_PASSWORD_REQUIRED challenge. It sets a password for user `jane@example.com`.

Command:

```
aws cognito-identity respond-to-auth-challenge --client-id 3n4b5urk1ft4fl3mg5e62d9ado  
--challenge-name NEW_PASSWORD_REQUIRED --challenge-responses  
USERNAME=jane@example.com,NEW_PASSWORD="password" --session "SESSION_TOKEN"
```

Output:

```
{  
    "ChallengeParameters": {},  
    "AuthenticationResult": {  
        "AccessToken": "ACCESS_TOKEN",  
        "ExpiresIn": 3600,  
        "TokenType": "Bearer",  
        "RefreshToken": "REFRESH_TOKEN",  
        "IdToken": "ID_TOKEN",  
        "NewDeviceMetadata": {  
            "DeviceKey": "us-west-2_fec070d2-fa88-424a-8ec8-b26d7198eb23",  
            "DeviceGroupKey": "-wt2ha1Zd"  
        }  
    }  
}
```

- For API details, see [RespondToAuthChallenge](#) in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const respondToAuthChallenge = ({  
    clientId,  
    username,  
    session,  
    userPoolId,  
    code,  
}) => {  
    const client = new CognitoIdentityProviderClient({});
```

```
const command = new RespondToAuthChallengeCommand({  
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,  
    ChallengeResponses: {  
        SOFTWARE_TOKEN_MFA_CODE: code,  
        USERNAME: username,  
    },  
    ClientId: clientId,  
    UserPoolId: userPoolId,  
    Session: session,  
});  
  
return client.send(command);  
};
```

- For API details, see [RespondToAuthChallenge](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Sign in with a tracked device. To complete sign-in, the client must respond correctly to Secure Remote Password (SRP) challenges.

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        """
```

```
:param client_secret: The client secret, if the client has a secret.  
"""  
self.cognito_idp_client = cognito_idp_client  
self.user_pool_id = user_pool_id  
self.client_id = client_id  
self.client_secret = client_secret  
  
def sign_in_with_tracked_device(  
    self,  
    user_name,  
    password,  
    device_key,  
    device_group_key,  
    device_password,  
    aws_srp,  
):  
    """  
    Signs in to Amazon Cognito as a user who has a tracked device. Signing in  
    with a tracked device lets a user sign in without entering a new MFA  
    code.  

```

Signing in with a tracked device requires that the client respond to the SRP protocol. The scenario associated with this example uses the warrant package to help with SRP calculations.

For more information on SRP, see https://en.wikipedia.org/wiki/Secure_Remote_Password_protocol.

```
:param user_name: The user that is associated with the device.  
:param password: The user's password.  
:param device_key: The key of a tracked device.  
:param device_group_key: The group key of a tracked device.  
:param device_password: The password that is associated with the device.  
:param aws_srp: A class that helps with SRP calculations. The scenario  
    associated with this example uses the warrant package.  
:return: The result of the authentication. When successful, this contains  
an  
    access token for the user.  
"""  
try:  
    srp_helper = aws_srp.AWSSRP(
```

```
        username=user_name,
        password=device_password,
        pool_id="_",
        client_id=self.client_id,
        client_secret=None,
        client=self.cognito_idp_client,
    )

response_init = self.cognito_idp_client.initiate_auth(
    ClientId=self.client_id,
    AuthFlow="USER_PASSWORD_AUTH",
    AuthParameters={
        "USERNAME": user_name,
        "PASSWORD": password,
        "DEVICE_KEY": device_key,
    },
)
if response_init["ChallengeName"] != "DEVICE_SRP_AUTH":
    raise RuntimeError(
        f"Expected DEVICE_SRP_AUTH challenge but got "
        f"{response_init['ChallengeName']}."
    )

auth_params = srp_helper.get_auth_params()
auth_params["DEVICE_KEY"] = device_key
response_auth = self.cognito_idp_client.respond_to_auth_challenge(
    ClientId=self.client_id,
    ChallengeName="DEVICE_SRP_AUTH",
    ChallengeResponses=auth_params,
)
if response_auth["ChallengeName"] != "DEVICE_PASSWORD_VERIFIER":
    raise RuntimeError(
        f"Expected DEVICE_PASSWORD_VERIFIER challenge but got "
        f"{response_init['ChallengeName']}."
    )

challenge_params = response_auth["ChallengeParameters"]
challenge_params["USER_ID_FOR_SRP"] = device_group_key + device_key
cr = srp_helper.process_challenge(challenge_params, {"USERNAME": user_name})
cr["USERNAME"] = user_name
cr["DEVICE_KEY"] = device_key
response_verifier =
self.cognito_idp_client.respond_to_auth_challenge(
```

```
        ClientId=self.client_id,
        ChallengeName="DEVICE_PASSWORD_VERIFIER",
        ChallengeResponses=cr,
    )
    auth_tokens = response_verifier["AuthenticationResult"]
except ClientError as err:
    logger.error(
        "Couldn't start client sign in for %s. Here's why: %s: %s",
        user_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return auth_tokens
```

- For API details, see [RespondToAuthChallenge](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use SignUp with an AWS SDK or CLI

The following code examples show how to use SignUp.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Automatically confirm known users with a Lambda function](#)
- [Automatically migrate known users with a Lambda function](#)
- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Sign up a new user.
/// </summary>
/// <param name="clientId">The client Id of the application.</param>
/// <param name="userName">The username to use.</param>
/// <param name="password">The user's password.</param>
/// <param name="email">The email address of the user.</param>
/// <returns>A Boolean value indicating whether the user was confirmed.</returns>
public async Task<bool> SignUpAsync(string clientId, string userName, string password, string email)
{
    var userAttrs = new AttributeType
    {
        Name = "email",
        Value = email,
    };

    var userAttrsList = new List<AttributeType>();

    userAttrsList.Add(userAttrs);

    var signUpRequest = new SignUpRequest
    {
        UserAttributes = userAttrsList,
        Username = userName,
        ClientId = clientId,
        Password = password
    };

    var response = await _cognitoService.SignUpAsync(signUpRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
```

```
}
```

- For API details, see [SignUp](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

Aws::CognitoIdentityProvider::Model::SignUpRequest request;
request.AddUserAttributes(
    Aws::CognitoIdentityProvider::Model::AttributeType().WithName(
        "email").WithValue(email));
request.SetUsername(userName);
request.SetPassword(password);
request.SetClientId(clientID);
Aws::CognitoIdentityProvider::Model::SignUpOutcome outcome =
    client.SignUp(request);

if (outcome.IsSuccess()) {
    std::cout << "The signup request for " << userName << " was
successful."
    << std::endl;
}
else if (outcome.GetError().GetErrorCode() ==

Aws::CognitoIdentityProvider::CognitoIdentityProviderErrors::USERNAME_EXISTS) {
    std::cout
```

```
        << "The username already exists. Please enter a different
username."
        << std::endl;
    userExists = true;
}
else {
    std::cerr << "Error with CognitoIdentityProvider::SignUpRequest. "
    << outcome.GetError().GetMessage()
    << std::endl;
    return false;
}
```

- For API details, see [SignUp](#) in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To sign up a user

This example signs up `jane@example.com`.

Command:

```
aws cognito-identity sign-up --client-id 3n4b5urk1ft4fl3mg5e62d9ado --
username jane@example.com --password PASSWORD --user-attributes
Name="email",Value="jane@example.com" Name="name",Value="Jane"
```

Output:

```
{
    "UserConfirmed": false,
    "UserSub": "e04d60a6-45dc-441c-a40b-e25a787d4862"
}
```

- For API details, see [SignUp](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
string, password string, userEmail string) (bool, error) {
    confirmed := false
    output, err := actor.CognitoClient.SignUp(ctx,
        &cognitoidentityprovider.SignUpInput{
            ClientId: aws.String(clientId),
            Password: aws.String(password),
            Username: aws.String(userName),
            UserAttributes: []types.AttributeType{
                {Name: aws.String("email"), Value: aws.String(userEmail)},
            },
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
```

```
if errors.As(err, &invalidPassword) {
    log.Println(*invalidPassword.Message)
} else {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
}
} else {
    confirmed = output.UserConfirmed
}
return confirmed, err
}
```

- For API details, see [SignUp](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
public static void signUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String userName,
                           String password, String email) {
    AttributeType userAttrs = AttributeType.builder()
        .name("email")
        .value(email)
        .build();

    List<AttributeType> userAttrsList = new ArrayList<>();
    userAttrsList.add(userAttrs);
    try {
        SignUpRequest signUpRequest = SignUpRequest.builder()
            .userAttributes(userAttrsList)
            .username(userName)
            .clientId(clientId)
            .password(password)
            .build();
```

```
        identityProviderClient.signUp(signUpRequest);
        System.out.println("User has been signed up ");

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [SignUp](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const signUp = ({ clientId, username, password, email }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new SignUpCommand({
        ClientId: clientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
    });

    return client.send(command);
};
```

- For API details, see [SignUp](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
suspend fun signUp(
    clientIdVal: String?,
    userNameVal: String?,
    passwordVal: String?,
    emailVal: String?,
) {
    val userAttrs =
        AttributeType {
            name = "email"
            value = emailVal
        }

    val userAttrsList = mutableListOf<AttributeType>()
    userAttrsList.add(userAttrs)
    val signUpRequest =
        SignUpRequest {
            userAttributes = userAttrsList
            username = userNameVal
            clientId = clientIdVal
            password = passwordVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        identityProviderClient.signUp(signUpRequest)
        println("User has been signed up")
    }
}
```

- For API details, see [SignUp](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
    def sign_up_user(self, user_name, password, user_email):  
        """  
        Signs up a new user with Amazon Cognito. This action prompts Amazon  
        Cognito  
        to send an email to the specified email address. The email contains a  
        code that  
        can be used to confirm the user.  
  
        When the user already exists, the user status is checked to determine  
        whether  
        the user has been confirmed.  
  
        :param user_name: The user name that identifies the new user.  
        """
```

```
:param password: The password for the new user.
:param user_email: The email address for the new user.
:return: True when the user is already confirmed with Amazon Cognito.
        Otherwise, false.
"""
try:
    kwargs = {
        "ClientId": self.client_id,
        "Username": user_name,
        "Password": password,
        "UserAttributes": [{"Name": "email", "Value": user_email}],
    }
    if self.client_secret is not None:
        kwargs["SecretHash"] = self._secret_hash(user_name)
    response = self.cognito_idp_client.sign_up(**kwargs)
    confirmed = response["UserConfirmed"]
except ClientError as err:
    if err.response["Error"]["Code"] == "UsernameExistsException":
        response = self.cognito_idp_client.admin_get_user(
            UserPoolId=self.user_pool_id, Username=user_name
        )
        logger.warning(
            "User %s exists and is %s.", user_name,
            response["UserStatus"]
        )
        confirmed = response["UserStatus"] == "CONFIRMED"
    else:
        logger.error(
            "Couldn't sign up %s. Here's why: %s: %s",
            user_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
return confirmed
```

- For API details, see [SignUp](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateUserPool` with an AWS SDK or CLI

The following code examples show how to use `UpdateUserPool`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- [Automatically confirm known users with a Lambda function](#)
- [Automatically migrate known users with a Lambda function](#)
- [Write custom activity data with a Lambda function after Amazon Cognito user authentication](#)

CLI

AWS CLI

To update a user pool

The following `update-user-pool` example modifies a user pool with example syntax for each of the available configuration options. To update a user pool, you must specify all previously-configured options or they will reset to a default value.

```
aws cognito-identity update-user-pool --user-pool-id us-west-2_EXAMPLE \
    --policies PasswordPolicy=
\{MinimumLength=6,RequireUppercase=true,RequireLowercase=true,RequireNumbers=true,Require
 \
    --deletion-protection ACTIVE \
    --lambda-config PreSignUp="arn:aws:lambda:us-
west-2:123456789012:function:cognito-test-presignup-
function",PreTokenGeneration="arn:aws:lambda:us-
west-2:123456789012:function:cognito-test-pretoken-function" \
    --auto-verified-attributes "phone_number" "email" \
    --verification-message-template \{"SmsMessage\":\\"Your code is
{####}\",\"EmailMessage\":\\"Your code is {####}\",\"EmailSubject\":\\"Your
verification code\",\"EmailMessageByLink\":\\"Click {##here##} to verify
your email address.\",\"EmailSubjectByLink\":\\"Your verification link\",
\"DefaultEmailOption\":\"CONFIRM_WITH_LINK\"\} \
    --sms-authentication-message "Your code is {####}" \
```

```
--user-attribute-update-settings
AttributesRequireVerificationBeforeUpdate="email","phone_number" \
--mfa-configuration "OPTIONAL" \
--device-
configuration ChallengeRequiredOnNewDevice=true,DeviceOnlyRememberedOnUserPrompt=true
\
--email-configuration SourceArn="arn:aws:ses:us-
west-2:123456789012:identity/admin@example.com",ReplyToEmailAddress="admin
+noreply@example.com",EmailSendingAccount=DEVELOPER,From="admin@amazon.com",Configuration
configuration-set" \
--sms-configuration SnsCallerArn="arn:aws:iam::123456789012:role/service-
role/SNS-SMS-Role",ExternalId="12345",SnsRegion="us-west-2" \
--admin-create-user-config
AllowAdminCreateUserOnly=false,InviteMessageTemplate=\{SMSMessage="""Welcome
{username}. Your confirmation code is {####}""",EmailMessage="""Welcome
{username}. Your confirmation code is {####}""",EmailSubject="""Welcome to
MyMobileGame"""\} \
--user-pool-tags "Function="MyMobileGame","Developers"="Berlin" \
--admin-create-user-config
AllowAdminCreateUserOnly=false,InviteMessageTemplate=\{SMSMessage="""Welcome
{username}. Your confirmation code is {####}""",EmailMessage="""Welcome
{username}. Your confirmation code is {####}""",EmailSubject="""Welcome to
MyMobileGame"""\} \
--user-pool-add-ons AdvancedSecurityMode="AUDIT" \
--account-recovery-setting RecoveryMechanisms=
\[\{Priority=1,Name="verified_email"\},
\{Priority=2,Name="verified_phone_number"\}\]
```

This command produces no output.

For more information, see [Updating user pool configuration](#) in the *Amazon Cognito Developer Guide*.

- For API details, see [UpdateUserPool](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger      Trigger
    HandlerArn  *string
}
```

```
// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
    string, triggers ...TriggerInfo) error {
    output, err := actor.CognitoClient.DescribeUserPool(ctx,
        &cognitoidentityprovider.DescribeUserPoolInput{
            UserPoolId: aws.String(userPoolId),
        })
    if err != nil {
        log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
            userPoolId, err)
        return err
    }
    lambdaConfig := output.UserPool.LambdaConfig
    for _, trigger := range triggers {
        switch trigger.Trigger {
        case PreSignUp:
            lambdaConfig.PreSignUp = trigger.HandlerArn
        case UserMigration:
            lambdaConfig.UserMigration = trigger.HandlerArn
        case PostAuthentication:
            lambdaConfig.PostAuthentication = trigger.HandlerArn
        }
    }
    _, err = actor.CognitoClient.UpdateUserPool(ctx,
        &cognitoidentityprovider.UpdateUserPoolInput{
            UserPoolId: aws.String(userPoolId),
            LambdaConfig: lambdaConfig,
        })
    if err != nil {
        log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
    }
    return err
}
```

- For API details, see [UpdateUserPool](#) in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/**  
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool  
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config  
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}  
 */  
export const addPreSignUpHandler = async ({  
    region,  
    userPoolId,  
    handlerArn,  
}) => {  
    try {  
        const cognitoClient = new CognitoIdentityProviderClient({  
            region,  
        });  
  
        const command = new UpdateUserPoolCommand({  
            UserPoolId: userPoolId,  
            LambdaConfig: {  
                PreSignUp: handlerArn,  
            },  
        });  
  
        const response = await cognitoClient.send(command);  
        return [response, null];  
    } catch (err) {  
        return [null, err];  
    }  
};
```

- For API details, see [UpdateUserPool](#) in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use VerifySoftwareToken with an AWS SDK

The following code examples show how to use VerifySoftwareToken.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Sign up a user with a user pool that requires MFA](#)

.NET

AWS SDK for .NET

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/// <summary>
/// Verify the TOTP and register for MFA.
/// </summary>
/// <param name="session">The name of the session.</param>
/// <param name="code">The MFA code.</param>
/// <returns>The status of the software token.</returns>
public async Task<VerifySoftwareTokenType>
    VerifySoftwareTokenAsync(string session, string code)
{
    var tokenRequest = new VerifySoftwareTokenRequest
    {
        UserCode = code,
        Session = session,
    };

    var verifyResponse = await
        _cognitoService.VerifySoftwareTokenAsync(tokenRequest);
```

```
        return verifyResponse.Status;
    }
```

- For API details, see [VerifySoftwareToken](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenRequest request;
request.SetUserCode(userCode);
request.SetSession(session);

Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenOutcome outcome =
    client.VerifySoftwareToken(request);

if (outcome.IsSuccess()) {
    std::cout << "Verification of the code was successful."
        << std::endl;
    session = outcome.GetResult().GetSession();
}
else {
    std::cerr << "Error with
CognitoIdentityProvider::VerifySoftwareToken. "
        << outcome.GetError().GetMessage()
        << std::endl;
}

return false;
```

```
}
```

- For API details, see [VerifySoftwareToken](#) in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Verify the TOTP and register for MFA.
public static void verifyTOTP(CognitoIdentityProviderClient
identityProviderClient, String session, String code) {
    try {
        VerifySoftwareTokenRequest tokenRequest =
VerifySoftwareTokenRequest.builder()
            .userCode(code)
            .session(session)
            .build();

        VerifySoftwareTokenResponse verifyResponse =
identityProviderClient.verifySoftwareToken(tokenRequest);
        System.out.println("The status of the token is " +
verifyResponse.statusAsString());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- For API details, see [VerifySoftwareToken](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
const verifySoftwareToken = (totp) => {
    const client = new CognitoIdentityProviderClient({});

    // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
    const session = process.env.SESSION;

    if (!session) {
        throw new Error(
            "Missing a valid Session. Did you run 'admin-initiate-auth?'",
        );
    }

    const command = new VerifySoftwareTokenCommand({
        Session: session,
        UserCode: totp,
    });

    return client.send(command);
};
```

- For API details, see [VerifySoftwareToken](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// Verify the TOTP and register for MFA.  
suspend fun verifyTOTP(  
    sessionVal: String?,  
    codeVal: String?,  
) {  
    val tokenRequest =  
        VerifySoftwareTokenRequest {  
            userCode = codeVal  
            session = sessionVal  
        }  
  
    CognitoIdentityProviderClient { region = "us-east-1" }.use  
{ identityProviderClient ->  
    val verifyResponse =  
    identityProviderClient.verifySoftwareToken(tokenRequest)  
    println("The status of the token is ${verifyResponse.status}")  
}  
}
```

- For API details, see [VerifySoftwareToken](#) in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
  
    def verify_mfa(self, session, user_code):  
        """  
        Verify a new MFA application that is associated with a user.  
  
        :param session: Session information returned from a previous call to  
        initiate  
                      authentication.  
        :param user_code: A code generated by the associated MFA application.  
        :return: Status that indicates whether the MFA application is verified.  
        """  
        try:  
            response = self.cognito_idp_client.verify_software_token(  
                Session=session, UserCode=user_code  
            )  
        except ClientError as err:  
            logger.error(  
                "Couldn't verify MFA. Here's why: %s: %s",  
                err.response["Error"]["Code"],  
                err.response["Error"]["Message"],  
            )  
            raise  
        else:  
            response.pop("ResponseMetadata", None)  
        return response
```

- For API details, see [VerifySoftwareToken](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Cognito Identity Provider using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Cognito Identity Provider with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon Cognito Identity Provider or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- [Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK](#)
- [Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK](#)
- [Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK](#)
- [Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK](#)

Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK

The following code examples show how to automatically confirm known Amazon Cognito users with a Lambda function.

- Configure a user pool to call a Lambda function for the PreSignUp trigger.
- Sign up a user with Amazon Cognito.
- The Lambda function scans a DynamoDB table and automatically confirms known users.
- Sign in as the new user, then clean up resources.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
import (
    "context"
    "errors"
    "log"
    "strings"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// AutoConfirm separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type AutoConfirm struct {
    helper      IScenarioHelper
    questioner  demotools.IQuestioner
    resources   Resources
    cognitoActor *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) AutoConfirm {
    scenario := AutoConfirm{
        helper:      helper,
        questioner:  questioner,
        resources:   Resources{},
    }
}
```

```
cognitoActor: &actions.CognitoActions{CognitoClient:  
cognitoidentityprovider.NewFromConfig(sdkConfig)},  
}  
scenario.resources.init(scenario.cognitoActor, questioner)  
return scenario  
}  
  
// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the  
PreSignUp trigger.  
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId  
string, functionArn string) {  
log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from  
Cognito.\n" +  
"This trigger happens when a user signs up, and lets your function take action  
before the main Cognito\n" +  
"sign up processing occurs.\n")  
err := runner.cognitoActor.UpdateTriggers(  
ctx, userPoolId,  
actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:  
aws.String(functionArn)})  
if err != nil {  
panic(err)  
}  
log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp  
trigger.\n",  
functionArn, userPoolId)  
}  
  
// SignUpUser signs up a user from the known user table with a password you  
specify.  
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,  
usersTable string) (string, string) {  
log.Println("Let's sign up a user to your Cognito user pool. When the user's  
email matches an email in the\n" +  
"DynamoDB known users table, it is automatically verified and the user is  
confirmed.")  
  
knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)  
if err != nil {  
panic(err)  
}  
userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",  
knownUsers.UserNameList())  
user := knownUsers.Users[userChoice]
```

```
var signedUp bool
var userConfirmed bool
password := runner.questioner.AskPassword("Enter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
"(the password will not display as you type):", 8)
for !signedUp {
    log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
    user.UserEmail)
    userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
    password, user.UserEmail)
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            password = runner.questioner.AskPassword("Enter another password:", 8)
        } else {
            panic(err)
        }
    } else {
        signedUp = true
    }
}
log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

log.Println(strings.Repeat("-", 88))

return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string,
userName string, password string) string {
runner.questioner.Ask("Press Enter when you're ready to continue.")
log.Printf("Let's sign in as %v...\n", userName)
authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
if err != nil {
    panic(err)
}
log.Printf("Successfully signed in. Your access token starts with: %v...\n",
(*authResult.AccessToken)[:10])
log.Println(strings.Repeat("-", 88))
return *authResult.AccessToken
}
```

```
// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup(ctx)
        }
    }()
}

log.Println(strings.Repeat("-", 88))
log.Printf("Welcome\n")

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]
runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])

runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
    stackOutputs["AutoConfirmFunctionArn"])
runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
    stackOutputs["TableName"])
runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
    runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

runner.resources.Cleanup(ctx)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}
```

Handle the PreSignUp trigger with a Lambda function.

```
import (
```

```
"context"
"log"
"os"

"github.com/aws/aws-lambda-go/events"
"github.com/aws/aws-lambda-go/lambda"
"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/feature/dynamodb/AttributeValue"
"github.com/aws/aws-sdk-go-v2/service/dynamodb"
dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
    userEmail, err := AttributeValue.Marshal(user.UserEmail)
    if err != nil {
        panic(err)
    }
    return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
// DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
error) {
    log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
event.UserName)
    if event.TriggerSource != "PreSignUp_SignUp" {
```

```
// Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the
// response from this handler.
return event, nil
}
tableName := os.Getenv(TABLE_NAME)
user := UserInfo{
    UserEmail: event.Request.UserAttributes["email"],
}
log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
    Key:      user.GetKey(),
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Error looking up email %v.\n", user.UserEmail)
    return event, err
}
if output.Item == nil {
    log.Printf("Email %v not found. Email verification is required.\n",
    user.UserEmail)
    return event, err
}

err = attributevalue.UnmarshalMap(output.Item, &user)
if err != nil {
    log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
    return event, err
}

if user.UserName != event.UserName {
    log.Printf("UserEmail %v found, but stored UserName '%v' does not match
supplied UserName '%v'. Verification is required.\n",
    user.UserEmail, user.UserName, event.UserName)
} else {
    log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.
\n", user.UserEmail, user.UserName)
    event.Response.AutoConfirmUser = true
    event.Response.AutoVerifyEmail = true
}

return event, err
}

func main() {
```

```
ctx := context.Background()
sdkConfig, err := config.LoadDefaultConfig(ctx)
if err != nil {
    log.Panicln(err)
}
h := handler{
    dynamoClient: dynamodb.NewFromConfig(sdkConfig),
}
lambda.Start(h.HandleRequest)
}
```

Create a struct that performs common tasks.

```
import (
    "context"
    "log"
    "strings"
    "time"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
        error)
    PopulateUserTable(ctx context.Context, tableName string)
    GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
    AddKnownUser(ctx context.Context, tableName string, user actions.User)
    ListRecentLogEvents(ctx context.Context, functionName string)
}
```

```
// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor    *actions.CloudFormationActions
    cwlActor    *actions.CloudWatchLogsActions
    isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner) ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
            dynamodb.NewFromConfig(sdkConfig)},
        cfnActor:    &actions.CloudFormationActions{CfnClient:
            cloudformation.NewFromConfig(sdkConfig)},
        cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
            cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
// structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
    string) (actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
    string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
    this example.\n", tableName)
```

```
err := helper.dynamoActor.PopulateTable(ctx, tableName)
if err != nil {
    panic(err)
}
}

// GetKnownUsers gets the users from the known users table in a structured
// format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
(actions.UserList, error) {
    knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
            tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
    user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
    table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(ctx, tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
    functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
    your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
```

```
events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
*logStream.LogStreamName, 10)
if err != nil {
    panic(err)
}
for _, event := range events {
    log.Printf("\t%v", *event.Message)
}
log.Println(strings.Repeat("-", 88))
}
```

Create a struct that wraps Amazon Cognito actions.

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
```

```
Trigger      Trigger
HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
string, triggers ...TriggerInfo) error {
output, err := actor.CognitoClient.DescribeUserPool(ctx,
&cognitoidentityprovider.DescribeUserPoolInput{
    UserPoolId: aws.String(userPoolId),
})
if err != nil {
    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
userPoolId, err)
    return err
}
lambdaConfig := output.UserPool.LambdaConfig
for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
        lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
        lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
        lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
}
_, err = actor.CognitoClient.UpdateUserPool(ctx,
&cognitoidentityprovider.UpdateUserPoolInput{
    UserPoolId:    aws.String(userPoolId),
    LambdaConfig: lambdaConfig,
})
if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
}
return err
}

// SignUp signs up a user with Amazon Cognito.
```

```
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName string, password string, userEmail string) (bool, error) {
    confirmed := false
    output, err := actor.CognitoClient.SignUp(ctx,
        &cognitoIdentityProvider.SignUpInput{
            ClientId: aws.String(clientId),
            Password: aws.String(password),
            Username: aws.String(userName),
            UserAttributes: []types.AttributeType{
                {Name: aws.String("email"), Value: aws.String(userEmail)},
            },
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
        }
    } else {
        confirmed = output.UserConfirmed
    }
    return confirmed, err
}
```

```
// SignIn signs in a user to Amazon Cognito using a username and password authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName string, password string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(ctx,
        &cognitoIdentityProvider.InitiateAuthInput{
            AuthFlow:      "USER_PASSWORD_AUTH",
            ClientId:     aws.String(clientId),
            AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
        })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    }
}
```

```
    }
} else {
    authResult = output.AuthenticationResult
}
return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoIdentityProvider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
        &cognitoIdentityProvider.ConfirmForgotPasswordInput{
            ClientId:      aws.String(clientId),
            ConfirmationCode: aws.String(code),
            Password:      aws.String(password),
            Username:      aws.String(userName),
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
    }
```

```
    log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
}
return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
  string) error {
_, err := actor.CognitoClient.DeleteUser(ctx,
&cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
})
if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
}
return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
  string, userName string, userEmail string) error {
_, err := actor.CognitoClient.AdminCreateUser(ctx,
&cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:      aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
aws.String(userEmail)}},
})
if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
    log.Printf("User %v already exists in the user pool.", userName)
    err = nil
  } else {
    log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
}
```

```
}

return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
    string, userName string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
    &cognitoidentityprovider.AdminSetUserPasswordInput{
        Password: aws.String(password),
        UserPoolId: aws.String(userPoolId),
        Username: aws.String(userName),
        Permanent: true,
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
        }
    }
    return err
}
```

Create a struct that wraps DynamoDB actions.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodbattribute"
```

```
"github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
    UserName  string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:"",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId   string
    Time       string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
// strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
    error {
    var err error
    var item map[string]types.AttributeValue
```

```
var writeReqs []types.WriteRequest
for i := 1; i < 4; i++ {
    item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
    if err != nil {
        log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
        return err
    }
    writeReqs = append(writeReqs, types.WriteRequest{PutRequest: &types.PutRequest{Item: item}})
}
_, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
if err != nil {
    log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName, err)
}
return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList, error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User) error {
```

```
userItem, err := attributevalue.MarshalMap(user)
if err != nil {
    log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
}
_, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
    Item:      userItem,
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
}
return err
}
```

Create a struct that wraps CloudWatch Logs actions.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
    functionName string) (types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(ctx,
        &cloudwatchlogs.DescribeLogStreamsInput{
            Descending: aws.Bool(true),
            Limit:      aws.Int32(1),
            LogGroupName: aws.String(logGroupName),
    })
    if err != nil {
        return logStream, err
    }
    logStream = output.Streams[0]
    return logStream, nil
}
```

```
    OrderBy:      types.OrderByLastEventTime,
})
if err != nil {
    log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
    logGroupName, err)
} else {
    logStream = output.LogStreams[0]
}
return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
string, logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(ctx,
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
    logStreamName, err)
} else {
    events = output.Events
}
return events, err
}
```

Create a struct that wraps AWS CloudFormation actions.

```
import (
    "context"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
```

```
"github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
    string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(ctx,
        &cloudformation.DescribeStacksInput{
            StackName: aws.String(stackName),
        })
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
            stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}
```

Clean up resources.

```
import (
    "context"
    "log"
    "user_pools_and_lambda_triggers/actions"

    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
```

```
// cleanup when the example finishes.

type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()
}

wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
    "during this demo (y/n)?", "y")
if wantDelete {
    for _, accessToken := range resources.userAccessTokens {
        err := resources.cognitoActor.DeleteUser(ctx, accessToken)
        if err != nil {
            log.Println("Couldn't delete user during cleanup.")
            panic(err)
        }
        log.Println("Deleted user.")
    }
    triggerList := make([]actions.TriggerInfo, len(resources.triggers))
    for i := 0; i < len(resources.triggers); i++ {
```

```
    triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],  
HandlerArn: nil}  
  }  
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,  
triggerList...)  
  if err != nil {  
    log.Println("Couldn't update Cognito triggers during cleanup.")  
    panic(err)  
  }  
  log.Println("Removed Cognito triggers from user pool.")  
} else {  
  log.Println("Be sure to remove resources when you're done with them to avoid  
unexpected charges!")  
}  
}  
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [SignUp](#)
 - [UpdateUserPool](#)

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Configure an interactive "Scenario" run. The JavaScript (v3) examples share a Scenario runner to streamline complex examples. The complete source code is on GitHub.

```
import { AutoConfirm } from "./scenario-auto-confirm.js";  
  
/**
```

```
* The context is passed to every scenario. Scenario steps
* will modify the context.
*/
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
      authentication behavior.",
  });
}
```

This Scenario demonstrates auto-confirming a known user. It orchestrates the example steps.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 */
```

```
*   TableName?: string,
*   UserPoolClientId?: string,
*   UserPoolId?: string,
*   UserPoolArn?: string,
*   AutoConfirmHandlerArn?: string,
*   AutoConfirmHandlerName?: string
* }) State
*/
const greeting = new ScenarioOutput(
  "greeting",
  (/* @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the autoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);
const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);
const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);
const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/* @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  }
);
```

```
        },
    );

const logSetupSignUpTrigger = new ScenarioOutput(
    "logSetupSignUpTrigger",
    "Setting up the PreSignUp trigger for the Cognito User Pool.",
    { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
    "setupSignUpTrigger",
    async (** @type {State} */ state) => {
        const [_, err] = await addPreSignUpHandler({
            region: state.stackRegion,
            userPoolId: state.UserPoolId,
            handlerArn: state.AutoConfirmHandlerArn,
        });
        if (err) {
            state.errors.push(err);
        }
    },
    {
        skipWhen: skipWhenErrors,
    },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
    "logSetupSignUpTriggerComplete",
    (
        /** @type {State} */ state,
    ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
"${state.UserPoolId}".`,
    { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
    "selectedUser",
    "Select a user to sign up.",
    {
        type: "select",
        choices: (** @type {State} */ state) => state.users.map((u) => u.UserName),
        skipWhen: skipWhenErrors,
        default: (** @type {State} */ state) => state.users[0].UserName,
```

```
    },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
"${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);
```

```
const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });
    let [_, err] = await signUp(state.password);

    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (** @type {State} */ state) =>
    `${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (** @type {State} */ state) => {
```

```
console.log(
  "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
);
await wait(10);

const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
  functionName: state.AutoConfirmHandlerName,
  region: state.stackRegion,
});
if (logStreamErr) {
  state.errors.push(logStreamErr);
  return;
}

console.log(
  `Getting some recent events from log stream "${logStream.logStreamName}"`,
);
const [logEvents, logEventsErr] = await getLogEvents({
  functionName: state.AutoConfirmHandlerName,
  region: state.stackRegion,
  eventCount: 10,
  logStreamName: logStream.logStreamName,
});
if (logEventsErr) {
  state.errors.push(logEventsErr);
  return;
}

console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
},
{ skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);
const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
```

```
        clientId: state.UserPoolClientId,
        username: state.selectedUser,
        password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
        state.errors.push(new Error("Please reset your password."));
        return;
    }

    if (err) {
        state.errors.push(err);
        return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
},
{ skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
    "logSignInUserComplete",
    /** @type {State} */ state) =>
`Successfully signed in. Your access token starts with:
${state.token.slice(0, 11)}',
{ skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
    "confirmDeleteSignedInUser",
    "Do you want to delete the currently signed in user?",
    { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
    "deleteSignedInUser",
    async /** @type {State} */ state) => {
    const [_, err] = await deleteUser({
        region: state.stackRegion,
        accessToken: state.token,
    });

    if (err) {
        state.errors.push(err);
    }
}
```

```
        },
      ],
      {
        skipWhen: (/* @type {State} */ state) =>
          skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
      },
    );
  }

const logErrors = new ScenarioOutput(
  "logErrors",
  (/* @type {State} */ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/* @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
```

```
    signInUser,
    logSignInUserComplete,
    confirmDeleteSignedInUser,
    deleteSignedInUser,
    logCleanUpReminder,
    logErrors,
  ],
  context,
);
```

These are steps that are shared with other Scenarios.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
```

```
{ type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

A handler for the PreSignUp trigger with a Lambda function.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "./user-repository";
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user
      '${event.userName}'`,
    );
  }
}
```

```
);

if (!this.isPreSignUpTriggerSource(event)) {
    return event;
}

const eventEmail = this.getEventUserEmail(event);
console.log(`Looking up email ${eventEmail}.`);
const storedUserInfo =
    await this.userRepository.getUserInfoByEmail(eventEmail);

if (!storedUserInfo) {
    console.log(
        `Email ${eventEmail} not found. Email verification is required.`,
    );
    return event;
}

if (storedUserInfo.UserName !== event.userName) {
    console.log(
        `UserEmail ${eventEmail} found, but stored UserName
        '${storedUserInfo.UserName}' does not match supplied UserName
        '${event.userName}'. Verification is required.`,
    );
} else {
    console.log(
        `UserEmail ${eventEmail} found with matching UserName
        ${storedUserInfo.UserName}. User is confirmed.`,
    );
    event.response.autoConfirmUser = true;
    event.response.autoVerifyEmail = true;
}
return event;
}

const createPreSignUpHandler = (): PreSignUpHandler => {
    const tableName = process.env.TABLE_NAME;
    if (!tableName) {
        throw new Error("TABLE_NAME environment variable is not set");
    }

    const userRepository = new DynamoDBUserRepository(tableName);
    return new PreSignUpHandler(userRepository);
```

```
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Module of CloudWatch Logs actions.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null, unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );
    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};
```

```
    }

};

/** 
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[]
 | null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module of Amazon Cognito actions.

```
import {
  AdminGetUserCommand,
```

```
CognitoIdentityProviderClient,
DeleteUserCommand,
InitiateAuthCommand,
SignUpCommand,
UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 */
```

```
*   password: string
* }) config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").SignUpCommandOutput | null, unknown]>}
*/
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 * flow.
 * @param {{ region: string, clientId: string, username: string, password: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
```

```
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
    )),
);
return [response, null];
} catch (err) {
    return [null, err];
}
};

/***
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").Admin GetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
        new Admin GetUserCommand({
            UserPoolId: userPoolId,
            Username: username,
        }),
    );
    return [response, null];
} catch (err) {
    return [null, err];
}
};

/***
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
        new DeleteUserCommand({ AccessToken: accessToken }),
```

```
    );
    return [response, null];
} catch (err) {
    return [null, err];
}
};
```

Module of DynamoDB actions.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }} config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput | null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
}
```

```
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [SignUp](#)
 - [UpdateUserPool](#)

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK

The following code example shows how to automatically migrate known Amazon Cognito users with a Lambda function.

- Configure a user pool to call a Lambda function for the `MigrateUser` trigger.
- Sign in to Amazon Cognito with a username and email that is not in the user pool.
- The Lambda function scans a DynamoDB table and automatically migrates known users to the user pool.
- Perform the forgot password flow to reset the password for the migrated user.
- Sign in as the new user, then clean up resources.

Go

SDK for Go V2

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
import (
    "context"
    "errors"
    "fmt"
    "log"
    "strings"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type MigrateUser struct {
    helper      IScenarioHelper
    questioner  demotools.IQuestioner
    resources   Resources
    cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) MigrateUser {
    scenario := MigrateUser{
        helper:      helper,
        questioner:  questioner,
        resources:   Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
            cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
// MigrateUser trigger.
```

```
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId string, functionArn string) {
    log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from Cognito.\n" +
        "This trigger happens when an unknown user signs in, and lets your function take action before Cognito\n" +
        "rejects the user.\n\n")
    err := runner.cognitoActor.UpdateTriggers(
        ctx, userPoolId,
        actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn: aws.String(functionArn)})
    if err != nil {
        panic(err)
    }
    log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser trigger.\n",
        functionArn, userPoolId)

    log.Println(strings.Repeat("-", 88))
}

// SignInUser adds a new user to the known users table and signs that user in to Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string, clientId string) (bool, actions.User) {
    log.Println("Let's sign in a user to your Cognito user pool. When the username and email matches an entry in the\n" +
        "DynamoDB known users table, the email is automatically verified and the user is migrated to the Cognito user pool.")

    user := actions.User{}
    user.UserName = runner.questioner.Ask("\nEnter a username:")
    user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This email will be used to confirm user migration\n" +
        "during this example:")

    runner.helper.AddKnownUser(ctx, usersTable, user)

    var err error
    var resetRequired *types.PasswordResetRequiredException
    var authResult *types.AuthenticationResultType
    signedIn := false
    for !signedIn && resetRequired == nil {
```

```
    log.Printf("Signing in to Cognito as user '%v'. The expected result is a\nPasswordResetRequiredException.\n\n", user.UserName)\n    authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")\n    if err != nil {\n        if errors.As(err, &resetRequired) {\n            log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the\nDynamoDB known users table.\n"+\n                "User migration is started and a password reset is required.",\n                user.UserName)\n        } else {\n            panic(err)\n        }\n    } else {\n        log.Printf("User '%v' successfully signed in. This is unexpected and probably\nmeans you have not\n"+\n                "cleaned up a previous run of this scenario, so the user exist in the Cognito\nuser pool.\n"+\n                "You can continue this example and select to clean up resources, or manually\nremove\n"+\n                "the user from your user pool and try again.", user.UserName)\n        runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,\n*authResult.AccessToken)\n        signedIn = true\n    }\n}\n\nlog.Println(strings.Repeat("-", 88))\nreturn resetRequired != nil, user\n}\n\n// ResetPassword starts a password recovery flow.\nfunc (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string,\nuser actions.User) {\n    wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user\nto Cognito, you must be able to receive a confirmation\n"+\n        "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")\n    if !wantCode {\n        log.Println("To complete this example and successfully migrate a user to\nCognito, you must enter an email\n" +\n            "you own that can receive a confirmation code.")\n        return\n    }\n    codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,\nuser.UserName)
```

```
if err != nil {
    panic(err)
}
log.Printf("\nA confirmation code has been sent to %v.",
*codeDelivery.Destination)
code := runner.questioner.Ask("Check your email and enter it here:")

confirmed := false
password := runner.questioner.AskPassword("\nEnter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
"(the password will not display as you type):", 8)
for !confirmed {
    log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
    err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
user.UserName, password)
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            password = runner.questioner.AskPassword("\nEnter another password:", 8)
        } else {
            panic(err)
        }
    } else {
        confirmed = true
    }
}
log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
log.Println("Signing in with your username and password...")
authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
password)
if err != nil {
    panic(err)
}
log.Printf("Successfully signed in. Your access token starts with: %v...\n",
(*authResult.AccessToken)[:10])
runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)

log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
    defer func() {
```

```
if r := recover(); r != nil {
    log.Println("Something went wrong with the demo.")
    runner.resources.Cleanup(ctx)
}
}()

log.Println(strings.Repeat("-", 88))
log.Printf("Welcome\n")

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]

runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
    stackOutputs["MigrateUserFunctionArn"])
runner.resources.triggers = append(runner.resources.triggers,
    actions.UserMigration)
resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
    stackOutputs["UserPoolClientId"])
if resetNeeded {
    runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
    runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
}

runner.resources.Cleanup(ctx)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}
```

Handle the `MigrateUser` trigger with a Lambda function.

```
import (
    "context"
    "log"
```

```
"os"

"github.com/aws/aws-lambda-go/events"
"github.com/aws/aws-lambda-go/lambda"
"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
"github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
"github.com/aws/aws-sdk-go-v2/service/dynamodb"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
    dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
// DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
events.CognitoEventUserPoolsMigrateUser)
(events.CognitoEventUserPoolsMigrateUser, error) {
    log.Printf("Received migrate trigger from %v for user '%v'", event.TriggerSource, event.UserName)
    if event.TriggerSource != "UserMigration_Authentication" {
        return event, nil
    }
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserName: event.UserName,
    }
    log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
    filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
    expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
    if err != nil {
```

```
    log.Printf("Error building expression to query for user '%v'.\n",
    user.UserName)
    return event, err
}
output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
    TableName:           aws.String(tableName),
    FilterExpression:   expr.Filter(),
    ExpressionAttributeNames: expr.Names(),
    ExpressionAttributeValues: expr.Values(),
})
if err != nil {
    log.Printf("Error looking up user '%v'.\n", user.UserName)
    return event, err
}
if len(output.Items) == 0 {
    log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
    return event, err
}

var users []UserInfo
err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
if err != nil {
    log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
    return event, err
}

user = users[0]
log.Printf("UserName '%v' found with email %v. User is migrated and must reset
password.\n", user.UserName, user.UserEmail)
event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes =
map[string]string{
    "email":           user.UserEmail,
    "email_verified": "true", // email_verified is required for the forgot password
flow.
}
event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus =
"RESET_REQUIRED"
event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

return event, err
}

func main() {
    ctx := context.Background()
```

```
sdkConfig, err := config.LoadDefaultConfig(ctx)
if err != nil {
    log.Panicln(err)
}
h := handler{
    dynamoClient: dynamodb.NewFromConfig(sdkConfig),
}
lambda.Start(h.HandleRequest)
}
```

Create a struct that performs common tasks.

```
import (
    "context"
    "log"
    "strings"
    "time"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
        error)
    PopulateUserTable(ctx context.Context, tableName string)
    GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
    AddKnownUser(ctx context.Context, tableName string, user actions.User)
    ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
```

```
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor    *actions.CloudFormationActions
    cwlActor    *actions.CloudWatchLogsActions
    isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner) ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
            dynamodb.NewFromConfig(sdkConfig)},
        cfnActor:    &actions.CloudFormationActions{CfnClient:
            cloudformation.NewFromConfig(sdkConfig)},
        cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
            cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
// structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
    string) (actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
    string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
    this example.\n", tableName)
    err := helper.dynamoActor.PopulateTable(ctx, tableName)
    if err != nil {
```

```
    panic(err)
}
}

// GetKnownUsers gets the users from the known users table in a structured
// format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
(actions.UserList, error) {
    knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
            tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
    user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
    table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(ctx, tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
    functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
    your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
    events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
        *logStream.LogStreamName, 10)
```

```
if err != nil {
    panic(err)
}
for _, event := range events {
    log.Printf("\t%v", *event.Message)
}
log.Println(strings.Repeat("-", 88))
}
```

Create a struct that wraps Amazon Cognito actions.

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
    Trigger    Trigger
    HandlerArn *string
}
```

```
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
string, triggers ...TriggerInfo) error {
output, err := actor.CognitoClient.DescribeUserPool(ctx,
&cognitoIdentityProvider.DescribeUserPoolInput{
    UserPoolId: aws.String(userPoolId),
})
if err != nil {
    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
userPoolId, err)
    return err
}
lambdaConfig := output.UserPool.LambdaConfig
for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
        lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
        lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
        lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
}
_, err = actor.CognitoClient.UpdateUserPool(ctx,
&cognitoIdentityProvider.UpdateUserPoolInput{
    UserPoolId:    aws.String(userPoolId),
    LambdaConfig: lambdaConfig,
})
if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
}
return err
}

// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
string, password string, userEmail string) (bool, error) {
confirmed := false
```

```
        output, err := actor.CognitoClient.SignUp(ctx,
&cognitoIdentityProvider.SignUpInput{
    ClientId: aws.String(clientId),
    Password: aws.String(password),
    Username: aws.String(userName),
    UserAttributes: []types.AttributeType{
        {Name: aws.String("email"), Value: aws.String(userEmail)},
    },
})
if err != nil {
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
        log.Println(*invalidPassword.Message)
    } else {
        log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
    }
} else {
    confirmed = output.UserConfirmed
}
return confirmed, err
}

// SignIn signs in a user to Amazon Cognito using a username and password
// authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
string, password string) (*types.AuthenticationResultType, error) {
var authResult *types.AuthenticationResultType
output, err := actor.CognitoClient.InitiateAuth(ctx,
&cognitoIdentityProvider.InitiateAuthInput{
    AuthFlow:      "USER_PASSWORD_AUTH",
    ClientId:     aws.String(clientId),
    AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
})
if err != nil {
    var resetRequired *types.PasswordResetRequiredException
    if errors.As(err, &resetRequired) {
        log.Println(*resetRequired.Message)
    } else {
        log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
    }
} else {
    authResult = output.AuthenticationResult
}
```

```
}

return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoIdentityProvider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
        &cognitoIdentityProvider.ConfirmForgotPasswordInput{
            ClientId:      aws.String(clientId),
            ConfirmationCode: aws.String(code),
            Password:      aws.String(password),
            Username:      aws.String(userName),
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
        }
    }
}
```

```
    return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
    string) error {
    _, err := actor.CognitoClient.DeleteUser(ctx,
    &cognitoidentityprovider.DeleteUserInput{
        AccessToken: aws.String(userAccessToken),
    })
    if err != nil {
        log.Printf("Couldn't delete user. Here's why: %v\n", err)
    }
    return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
// This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
    string, userName string, userEmail string) error {
    _, err := actor.CognitoClient.AdminCreateUser(ctx,
    &cognitoidentityprovider.AdminCreateUserInput{
        UserPoolId:     aws.String(userPoolId),
        Username:      aws.String(userName),
        MessageAction: types.MessageActionTypeSuppress,
        UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
            aws.String(userEmail)}},
    })
    if err != nil {
        var userExists *types.UsernameExistsException
        if errors.As(err, &userExists) {
            log.Printf("User %v already exists in the user pool.", userName)
            err = nil
        } else {
            log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
        }
    }
    return err
}
```

```
// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
    string, userName string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
    &cognitoidentityprovider.AdminSetUserPasswordInput{
        Password: aws.String(password),
        UserPoolId: aws.String(userPoolId),
        Username: aws.String(userName),
        Permanent: true,
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
            err)
        }
    }
    return err
}
```

Create a struct that wraps DynamoDB actions.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
    UserName  string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:"",omitempty"`
}
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId   string
    Time       string
}
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}
}

// UserNameList returns the usernames contained in a UserList as a list of
// strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
    error {
    var err error
    var item map[string]types.AttributeValue
    var writeReqs []types.WriteRequest
    for i := 1; i < 4; i++ {
```

```
item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
if err != nil {
    log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
    return err
}
writeReqs = append(writeReqs, types.WriteRequest{PutRequest: &types.PutRequest{Item: item}})
}
_, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
if err != nil {
    log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName, err)
}
return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList, error) {
var userList UserList
output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
} else {
    err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
    if err != nil {
        log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
    }
}
return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User) error {
userItem, err := attributevalue.MarshalMap(user)
if err != nil {
```

```
    log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
}
_, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
    Item:      userItem,
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
}
return err
}
```

Create a struct that wraps CloudWatch Logs actions.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
    functionName string) (types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(ctx,
        &cloudwatchlogs.DescribeLogStreamsInput{
            Descending: aws.Bool(true),
            Limit:      aws.Int32(1),
            LogGroupName: aws.String(logGroupName),
            OrderBy:     types.OrderByLastEventTime,
        })
    if err != nil {
        log.Println("Error getting log streams: ", err)
        return logStream, err
    }
    if len(output.Streams) == 0 {
        log.Println("No log streams found for function: ", functionName)
        return logStream, nil
    }
    logStream = output.Streams[0]
    return logStream, nil
}
```

```
if err != nil {
    log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
    logGroupName, err)
} else {
    logStream = output.LogStreams[0]
}
return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
    string, logStreamName string, eventCount int32) (
    []types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(ctx,
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
    logStreamName, err)
} else {
    events = output.Events
}
return events, err
}
```

Create a struct that wraps AWS CloudFormation actions.

```
import (
    "context"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)
```

```
// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(ctx,
&cloudformation.DescribeStacksInput{
    StackName: aws.String(stackName),
})
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}
```

Clean up resources.

```
import (
    "context"
    "log"
    "user_pools_and_lambda_triggers/actions"

    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
```

```
userPoolId      string
userAccessTokens []string
triggers        []actions.Trigger

cognitoActor *actions.CognitoActions
questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
resources.userAccessTokens = []string{}
resources.triggers = []actions.Trigger{}
resources.cognitoActor = cognitoActor
resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
defer func() {
if r := recover(); r != nil {
log.Printf("Something went wrong during cleanup.\n%v\n", r)
log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
"that were created for this scenario.")
}
}()

wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
"during this demo (y/n)?", "y")
if wantDelete {
for _, accessToken := range resources.userAccessTokens {
err := resources.cognitoActor.DeleteUser(ctx, accessToken)
if err != nil {
log.Println("Couldn't delete user during cleanup.")
panic(err)
}
log.Println("Deleted user.")
}
triggerList := make([]actions.TriggerInfo, len(resources.triggers))
for i := 0; i < len(resources.triggers); i++ {
triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
HandlerArn: nil}
}
```

```
    err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
triggerList...)
if err != nil {
    log.Println("Couldn't update Cognito triggers during cleanup.")
    panic(err)
}
log.Println("Removed Cognito triggers from user pool.")
} else {
    log.Println("Be sure to remove resources when you're done with them to avoid
unexpected charges!")
}
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.

- [ConfirmForgotPassword](#)
- [DeleteUser](#)
- [ForgotPassword](#)
- [InitiateAuth](#)
- [SignUp](#)
- [UpdateUserPool](#)

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK

The following code examples show how to:

- Sign up and confirm a user with a username, password, and email address.
- Set up multi-factor authentication by associating an MFA application with the user.
- Sign in by using a password and an MFA code.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
namespace CognitoBasics;

public class CognitoBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Cognito.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCognitoIdentityProvider>()
                    .AddTransient<CognitoWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CognitoBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();
    }
}
```

```
var cognitoWrapper = host.Services.GetRequiredService<CognitoWrapper>();

Console.WriteLine(new string('-', 80));
UiMethods.DisplayOverview();
Console.WriteLine(new string('-', 80));

    // clientId - The app client Id value that you get from the AWS CDK
    script.
    var clientId = configuration["ClientId"]; // *** REPLACE WITH CLIENT ID
    VALUE FROM CDK SCRIPT";

    // poolId - The pool Id that you get from the AWS CDK script.
    var poolId = configuration["PoolId"]!; // *** REPLACE WITH POOL ID VALUE
    FROM CDK SCRIPT";
    var userName = configuration["UserName"];
    var password = configuration["Password"];
    var email = configuration["Email"];

    // If the username wasn't set in the configuration file,
    // get it from the user now.
    if (userName is null)
    {
        do
        {
            Console.Write("Username: ");
            userName = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(userName));
    }
    Console.WriteLine($"\\nUsername: {userName}");

    // If the password wasn't set in the configuration file,
    // get it from the user now.
    if (password is null)
    {
        do
        {
            Console.Write("Password: ");
            password = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(password));
    }

    // If the email address wasn't set in the configuration file,
```

```
// get it from the user now.
if (email is null)
{
    do
    {
        Console.Write("Email: ");
        email = Console.ReadLine();
    } while (string.IsNullOrEmpty(email));
}

// Now sign up the user.
Console.WriteLine($"\\nSigning up {userName} with email address:
{email}");
await cognitoWrapper.SignUpAsync(clientId, userName, password, email);

// Add the user to the user pool.
Console.WriteLine($"Adding {userName} to the user pool");
await cognitoWrapper.GetAdminUserAsync(userName, poolId);

UiMethods.DisplayTitle("Get confirmation code");
Console.WriteLine($"Conformation code sent to {userName}.");
Console.Write("Would you like to send a new code? (Y/N) ");
var answer = Console.ReadLine();

if (answer!.ToLower() == "y")
{
    await cognitoWrapper.ResendConfirmationCodeAsync(clientId, userName);
    Console.WriteLine("Sending a new confirmation code");
}

Console.Write("Enter confirmation code (from Email): ");
var code = Console.ReadLine();

await cognitoWrapper.ConfirmSignupAsync(clientId, code, userName);

UiMethods.DisplayTitle("Checking status");
Console.WriteLine($"Rechecking the status of {userName} in the user
pool");
await cognitoWrapper.GetAdminUserAsync(userName, poolId);

Console.WriteLine($"Setting up authenticator for {userName} in the user
pool");
var setupResponse = await cognitoWrapper.InitiateAuthAsync(clientId,
userName, password);
```

```
        var setupSession = await
cognitoWrapper.AssociateSoftwareTokenAsync(setupResponse.Session);
        Console.WriteLine("Enter the 6-digit code displayed in Google Authenticator:");
    });
    var setupCode = Console.ReadLine();

    var setupResult = await
cognitoWrapper.VerifySoftwareTokenAsync(setupSession, setupCode);
    Console.WriteLine($"Setup status: {setupResult}");

    Console.WriteLine($"Now logging in {userName} in the user pool");
    var authSession = await cognitoWrapper.AdminInitiateAuthAsync(clientId,
poolId, userName, password);

    Console.WriteLine("Enter a new 6-digit code displayed in Google
Authenticator: ");
    var authCode = Console.ReadLine();

    var authResult = await
cognitoWrapper.AdminRespondToAuthChallengeAsync(userName, clientId, authCode,
authSession, poolId);
    Console.WriteLine($"Authenticated and received access token:
{authResult.AccessToken}");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine("Cognito scenario is complete.");
    Console.WriteLine(new string('-', 80));
}

}

using System.Net;

namespace CognitoActions;

/// <summary>
/// Methods to perform Amazon Cognito Identity Provider actions.
/// </summary>
public class CognitoWrapper
{
    private readonly IAmazonCognitoIdentityProvider _cognitoService;

    /// <summary>
```

```
/// Constructor for the wrapper class containing Amazon Cognito actions.  
/// </summary>  
/// <param name="cognitoService">The Amazon Cognito client object.</param>  
public CognitoWrapper(IAmazonCognitoIdentityProvider cognitoService)  
{  
    _cognitoService = cognitoService;  
}  
  
/// <summary>  
/// List the Amazon Cognito user pools for an account.  
/// </summary>  
/// <returns>A list of UserPoolDescriptionType objects.</returns>  
public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()  
{  
    var userPools = new List<UserPoolDescriptionType>();  
  
    var userPoolsPaginator = _cognitoService.Paginator.ListUserPools(new  
ListUserPoolsRequest());  
  
    await foreach (var response in userPoolsPaginator.Responses)  
    {  
        userPools.AddRange(response.UserPools);  
    }  
  
    return userPools;  
}  
  
/// <summary>  
/// Get a list of users for the Amazon Cognito user pool.  
/// </summary>  
/// <param name="userPoolId">The user pool ID.</param>  
/// <returns>A list of users.</returns>  
public async Task<List<UserType>> ListUsersAsync(string userPoolId)  
{  
    var request = new ListUsersRequest  
    {  
        UserPoolId = userPoolId  
    };  
  
    var users = new List<UserType>();  
  
    var usersPaginator = _cognitoService.Paginator.ListUsers(request);  
    await foreach (var response in usersPaginator.Responses)
```

```
        }

        users.AddRange(response.Users);

    }

    return users;
}

///<summary>
/// Respond to an admin authentication challenge.
///</summary>
///<param name="userName">The name of the user.</param>
///<param name="clientId">The client ID.</param>
///<param name="mfaCode">The multi-factor authentication code.</param>
///<param name="session">The current application session.</param>
///<param name="clientId">The user pool ID.</param>
///<returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
    string mfaCode,
    string session,
    string userPoolId)
{
    Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

    var challengeResponses = new Dictionary<string, string>();
    challengeResponses.Add("USERNAME", userName);
    challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

    var respondToAuthChallengeRequest = new
AdminRespondToAuthChallengeRequest
    {
        ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
        ClientId = clientId,
        ChallengeResponses = challengeResponses,
        Session = session,
        UserPoolId = userPoolId,
    };

    var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
    Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
}
```

```
        return response.AuthenticationResult;
    }

    /// <summary>
    /// Verify the TOTP and register for MFA.
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
{
    var tokenRequest = new VerifySoftwareTokenRequest
    {
        UserCode = code,
        Session = session,
    };

    var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

    return verifyResponse.Status;
}

    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
{
    var softwareTokenRequest = new AssociateSoftwareTokenRequest
    {
        Session = session,
    };

    var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
    var secretCode = tokenResponse.SecretCode;

    Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");
}
```

```
        return tokenResponse.Session;
    }

    ///<summary>
    ///<summary>Initiate an admin auth request.
    ///</summary>
    ///<param name="clientId">The client ID to use.</param>
    ///<param name="userPoolId">The ID of the user pool.</param>
    ///<param name="userName">The username to authenticate.</param>
    ///<param name="password">The user's password.</param>
    ///<returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
{
    var authParameters = new Dictionary<string, string>();
    authParameters.Add("USERNAME", userName);
    authParameters.Add("PASSWORD", password);

    var request = new AdminInitiateAuthRequest
    {
        ClientId = clientId,
        UserPoolId = userPoolId,
        AuthParameters = authParameters,
        AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    };

    var response = await _cognitoService.AdminInitiateAuthAsync(request);
    return response.Session;
}

    ///<summary>
    ///<summary>Initiate authorization.
    ///</summary>
    ///<param name="clientId">The client Id of the application.</param>
    ///<param name="userName">The name of the user who is authenticating.</
param>
    ///<param name="password">The password for the user who is authenticating.</
param>
    ///<returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
string userName, string password)
{
```

```
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }

    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code,
string userName)
{
    var signUpRequest = new ConfirmSignUpRequest
    {
        ClientId = clientId,
        ConfirmationCode = code,
        Username = userName,
    };

    var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
    if (response.HttpStatusCode == HttpStatusCode.OK)
    {
        Console.WriteLine($"{userName} was confirmed");
        return true;
    }
    return false;
}
```

```
/// <summary>
/// Initiates and confirms tracking of the device.
/// </summary>
/// <param name="accessToken">The user's access token.</param>
/// <param name="deviceKey">The key of the device from Amazon Cognito.</param>
/// <param name="deviceName">The device name.</param>
/// <returns></returns>
public async Task<bool> ConfirmDeviceAsync(string accessToken, string deviceKey, string deviceName)
{
    var request = new ConfirmDeviceRequest
    {
        AccessToken = accessToken,
        DeviceKey = deviceKey,
        DeviceName = deviceName
    };

    var response = await _cognitoService.ConfirmDeviceAsync(request);
    return response.UserConfirmationNecessary;
}

/// <summary>
/// Send a new confirmation code to a user.
/// </summary>
/// <param name="clientId">The Id of the client application.</param>
/// <param name="userName">The username of user who will receive the code.</param>
/// <returns>The delivery details.</returns>
public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string clientId, string userName)
{
    var codeRequest = new ResendConfirmationCodeRequest
    {
        ClientId = clientId,
        Username = userName,
    };

    var response = await
_cognitoService.ResendConfirmationCodeAsync(codeRequest);
```

```
        Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }

    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with
    administrator access.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
    {
        Admin GetUserRequest userRequest = new Admin GetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.Admin GetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }

    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
password, string email)
    {
        var userAttrs = new AttributeType
        {
```

```
        Name = "email",
        Value = email,
    };

    var userAttrsList = new List<AttributeType>();

    userAttrsList.Add(userAttrs);

    var signUpRequest = new SignUpRequest
    {
        UserAttributes = userAttrsList,
        Username = userName,
        ClientId = clientId,
        Password = password
    };

    var response = await _cognitoService.SignUpAsync(signUpRequest);
    return response.HttpStatusCode == HttpStatusCode.OK;
}

}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.

- [Admin GetUser](#)
- [Admin Initiate Auth](#)
- [Admin Respond To Auth Challenge](#)
- [Associate Software Token](#)
- [Confirm Device](#)
- [Confirm Sign Up](#)
- [Initiate Auth](#)
- [List Users](#)
- [Resend Confirmation Code](#)
- [Respond To Auth Challenge](#)
- [Sign Up](#)
- [Verify Software Token](#)

C++

SDK for C++**Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

//! Scenario that adds a user to an Amazon Cognito user pool.
/*
\sa gettingStartedWithUserPools()
\param clientID: Client ID associated with an Amazon Cognito user pool.
\param userPoolID: An Amazon Cognito user pool ID.
\param clientConfig: Aws client configuration.
\return bool: Successful completion.
*/
bool AwsDoc::Cognito::gettingStartedWithUserPools(const Aws::String &clientID,
                                                   const Aws::String &userPoolID,
                                                   const
Aws::Client::ClientConfiguration &clientConfig) {
    printAsterisksLine();
    std::cout
        << "Welcome to the Amazon Cognito example scenario."
        << std::endl;
    printAsterisksLine();

    std::cout
        << "This scenario will add a user to an Amazon Cognito user pool."
        << std::endl;
    const Aws::String userName = askQuestion("Enter a new username: ");
    const Aws::String password = askQuestion("Enter a new password: ");
    const Aws::String email = askQuestion("Enter a valid email for the user: ");

    std::cout << "Signing up " << userName << std::endl;
```

```
Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);
bool userExists = false;
do {
    // 1. Add a user with a username, password, and email address.
    Aws::CognitoIdentityProvider::Model::SignUpRequest request;
    request.AddUserAttributes(
        Aws::CognitoIdentityProvider::Model::AttributeType().WithName(
            "email").WithValue(email));
    request.SetUsername(userName);
    request.SetPassword(password);
    request.SetClientId(clientID);
    Aws::CognitoIdentityProvider::Model::SignUpOutcome outcome =
        client.SignUp(request);

    if (outcome.IsSuccess()) {
        std::cout << "The signup request for " << userName << " was
successful."
        << std::endl;
    }
    else if (outcome.GetError().GetErrorCode() ==
        Aws::CognitoIdentityProvider::CognitoIdentityProviderErrors::USERNAME_EXISTS) {
        std::cout
            << "The username already exists. Please enter a different
username."
            << std::endl;
        userExists = true;
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::SignUpRequest. "
        << outcome.GetError().GetMessage()
        << std::endl;
        return false;
    }
} while (userExists);

printAsterisksLine();
std::cout << "Retrieving status of " << userName << " in the user pool."
<< std::endl;
// 2. Confirm that the user was added to the user pool.
if (!checkAdminUserStatus(userName, userPoolID, client)) {
    return false;
}
```

```
    std::cout << "A confirmation code was sent to " << email << "." << std::endl;

    bool resend = askYesNoQuestion("Would you like to send a new code? (y/n) ");
    if (resend) {
        // Request a resend of the confirmation code to the email address.
        (ResendConfirmationCode)
            Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeRequest
        request;
        request.SetUsername(userName);
        request.SetClientId(clientID);

        Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeOutcome
        outcome =
            client.ResendConfirmationCode(request);

        if (outcome.IsSuccess()) {
            std::cout
                << "CognitoIdentityProvider::ResendConfirmationCode was
successful."
                << std::endl;
        }
        else {
            std::cerr << "Error with
CognitoIdentityProvider::ResendConfirmationCode. "
                << outcome.GetError().GetMessage()
                << std::endl;
            return false;
        }
    }

    printAsterisksLine();

    {
        // 4. Send the confirmation code that's received in the email.
        (ConfirmSignUp)
            const Aws::String confirmationCode = askQuestion(
                "Enter the confirmation code that was emailed: ");
            Aws::CognitoIdentityProvider::Model::ConfirmSignUpRequest request;
            request.SetClientId(clientID);
            request.SetConfirmationCode(confirmationCode);
            request.SetUsername(userName);

            Aws::CognitoIdentityProvider::Model::ConfirmSignUpOutcome outcome =
```

```
        client.ConfirmSignUp(request);

        if (outcome.IsSuccess()) {
            std::cout << "ConfirmSignup was Successful."
                << std::endl;
        }
        else {
            std::cerr << "Error with CognitoIdentityProvider::ConfirmSignUp. "
                << outcome.GetError().GetMessage()
                << std::endl;
            return false;
        }
    }

    std::cout << "Rechecking the status of " << userName << " in the user pool."
        << std::endl;
    if (!checkAdminUserStatus(userName, userPoolID, client)) {
        return false;
    }

    printAsterisksLine();

    std::cout << "Initiating authorization using the username and password."
        << std::endl;

    Aws::String session;
    // 5. Initiate authorization with username and password. (AdminInitiateAuth)
    if (!adminInitiateAuthorization(clientID, userPoolID, userName, password,
session, client)) {
        return false;
    }

    printAsterisksLine();

    std::cout
        << "Starting setup of time-based one-time password (TOTP) multi-
factor authentication (MFA)."
        << std::endl;

    {
        // 6. Request a setup key for one-time password (TOTP)
        //     multi-factor authentication (MFA). (AssociateSoftwareToken)
        Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenRequest
request;
```

```
request.SetSession(session);

Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenOutcome
outcome =
    client.AssociateSoftwareToken(request);

if (outcome.IsSuccess()) {
    std::cout
        << "Enter this setup key into an authenticator app, for
example Google Authenticator."
        << std::endl;
    std::cout << "Setup key: " << outcome.GetResult().GetSecretCode()
        << std::endl;

#ifndef USING_QR
    printAsterisksLine();
    std::cout << "\nOr scan the QR code in the file '" << QR_CODE_PATH <<
    "."
        << std::endl;

    saveQRCode(std::string("otpauth://totp/") + userName + "?secret=" +
               outcome.GetResult().GetSecretCode());
#endif // USING_QR
    session = outcome.GetResult().GetSession();
}
else {
    std::cerr << "Error with
CognitoIdentityProvider::AssociateSoftwareToken. "
               << outcome.GetError().GetMessage()
               << std::endl;
    return false;
}
askQuestion("Type enter to continue...", alwaysTrueTest);

printAsterisksLine();

{
    Aws::String userCode = askQuestion(
        "Enter the 6 digit code displayed in the authenticator app: ");

    // 7. Send the MFA code copied from an authenticator app.
(VerifySoftwareToken)
    Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenRequest request;
    request.SetUserCode(userCode);
```

```
request.SetSession(session);

Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenOutcome outcome =
    client.VerifySoftwareToken(request);

if (outcome.IsSuccess()) {
    std::cout << "Verification of the code was successful."
        << std::endl;
    session = outcome.GetResult().GetSession();
}
else {
    std::cerr << "Error with
CognitoIdentityProvider::VerifySoftwareToken. "
        << outcome.GetError().GetMessage()
        << std::endl;
    return false;
}

printAsterisksLine();
std::cout << "You have completed the MFA authentication setup." << std::endl;
std::cout << "Now, sign in." << std::endl;

// 8. Initiate authorization again with username and password.
(AdminInitiateAuth)
if (!adminInitiateAuthorization(clientID, userPoolID, userName, password,
session, client)) {
    return false;
}

Aws::String accessToken;
{
    Aws::String mfaCode = askQuestion(
        "Re-enter the 6 digit code displayed in the authenticator app:
");

    // 9. Send a new MFA code copied from an authenticator app.
(AdminRespondToAuthChallenge)
    Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeRequest
request;
    request.AddChallengeResponses("USERNAME", userName);
    request.AddChallengeResponses("SOFTWARE_TOKEN_MFA_CODE", mfaCode);
    request.SetChallengeName(
```

```
Aws::CognitoIdentityProvider::Model::ChallengeNameType::SOFTWARE_TOKEN_MFA);
    request.SetClientId(clientID);
    request.SetUserPoolId(userPoolID);
    request.SetSession(session);

    Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeOutcome
outcome =
    client.AdminRespondToAuthChallenge(request);

    if (outcome.IsSuccess()) {
        std::cout << "Here is the response to the challenge.\n" <<
        outcome.GetResult().GetAuthenticationResult().Jsonize().View().WriteReadable()
            << std::endl;

        accessToken =
outcome.GetResult().GetAuthenticationResult().GetAccessToken();
    }
    else {
        std::cerr << "Error with
CognitoIdentityProvider::AdminRespondToAuthChallenge. "
            << outcome.GetError().GetMessage()
            << std::endl;
        return false;
    }

    std::cout << "You have successfully added a user to Amazon Cognito."
            << std::endl;
}

if (askYesNoQuestion("Would you like to delete the user that you just added?
(y/n) ")) {
    // 10. Delete the user that you just added. (DeleteUser)
    Aws::CognitoIdentityProvider::Model::DeleteUserRequest request;
    request.SetAccessToken(accessToken);

    Aws::CognitoIdentityProvider::Model::DeleteUserOutcome outcome =
        client.DeleteUser(request);

    if (outcome.IsSuccess()) {
        std::cout << "The user " << userName << " was deleted."
            << std::endl;
    }
}
```

```
        else {
            std::cerr << "Error with CognitoIdentityProvider::DeleteUser. "
            << outcome.GetError().GetMessage()
            << std::endl;
        }
    }

    return true;
}

//! Routine which checks the user status in an Amazon Cognito user pool.
/*!
\sa checkAdminUserStatus()
\param userName: A username.
\param userPoolID: An Amazon Cognito user pool ID.
\return bool: Successful completion.
*/
bool AwsDoc::Cognito::checkAdminUserStatus(const Aws::String &userName,
                                             const Aws::String &userPoolID,
                                             const
                                             Aws::CognitoIdentityProvider::CognitoIdentityProviderClient &client) {
    Aws::CognitoIdentityProvider::Model::Admin GetUserRequest request;
    request.SetUsername(userName);
    request.SetUserPoolId(userPoolID);

    Aws::CognitoIdentityProvider::Model::Admin GetUserOutcome outcome =
        client.Admin GetUser(request);

    if (outcome.IsSuccess()) {
        std::cout << "The status for " << userName << " is " <<

        Aws::CognitoIdentityProvider::Model::User StatusTypeMapper::GetNameForUserStatusType(
            outcome.GetResult(). GetUserStatus()) << std::endl;
        std::cout << "Enabled is " << outcome.GetResult().GetEnabled() <<
        std::endl;
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::Admin GetUser. "
        << outcome.GetError().GetMessage()
        << std::endl;
    }

    return outcome.IsSuccess();
}
```

```
///! Routine which starts authorization of an Amazon Cognito user.
///! This routine requires administrator credentials.
<*/
\sa adminInitiateAuthorization()
\param clientID: Client ID of tracked device.
\param userPoolID: An Amazon Cognito user pool ID.
\param userName: A username.
\param password: A password.
\param sessionResult: String to receive a session token.
\return bool: Successful completion.
*/
bool AwsDoc::Cognito::adminInitiateAuthorization(const Aws::String &clientID,
                                                    const Aws::String &userPoolID,
                                                    const Aws::String &userName,
                                                    const Aws::String &password,
                                                    Aws::String &sessionResult,
                                                    const
Aws::CognitoIdentityProvider::CognitoIdentityProviderClient &client) {
    Aws::CognitoIdentityProvider::Model::AdminInitiateAuthRequest request;
    request.SetClientId(clientID);
    request.SetUserPoolId(userPoolID);
    request.AddAuthParameters("USERNAME", userName);
    request.AddAuthParameters("PASSWORD", password);
    request.SetAuthFlow(
        Aws::CognitoIdentityProvider::Model::AuthFlowType::ADMIN_USER_PASSWORD_AUTH);

    Aws::CognitoIdentityProvider::Model::AdminInitiateAuthOutcome outcome =
        client.AdminInitiateAuth(request);

    if (outcome.IsSuccess()) {
        std::cout << "Call to AdminInitiateAuth was successful." << std::endl;
        sessionResult = outcome.GetResult().GetSession();
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::AdminInitiateAuth. "
              << outcome.GetError().GetMessage()
              << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
 - [Admin GetUser](#)
 - [Admin Initiate Auth](#)
 - [Admin Respond To Auth Challenge](#)
 - [Associate Software Token](#)
 - [Confirm Device](#)
 - [Confirm Sign Up](#)
 - [Initiate Auth](#)
 - [List Users](#)
 - [Resend Confirmation Code](#)
 - [Respond To Auth Challenge](#)
 - [Sign Up](#)
 - [Verify Software Token](#)

Java

SDK for Java 2.x

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import
software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.Admin GetUserRequest;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.Admin GetUserResponse;
import
software.amazon.awssdk.services.cognitoidentityprovider.model.Admin Initiate AuthRequest;
```

```
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminInitiateAuthResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminRespondToAuthChallenge;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminRespondToAuthChallengeRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AssociateSoftwareTokenRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AssociateSoftwareTokenResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AttributeType;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AuthFlowType;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ChallengeNameType;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ConfirmSignUpRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ResendConfirmationCodeRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ResendConfirmationCodeResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.SignUpRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.VerifySoftwareTokenRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.VerifySoftwareTokenResponse;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
*
* TIP: To set up the required user pool, run the AWS Cloud Development Kit (AWS CDK) script provided in this GitHub repo at
* resources/cdk/cognito_scenario_user_pool_with_mfa.
*
* This code example performs the following operations:
*
* 1. Invokes the signUp method to sign up a user.
* 2. Invokes the adminGetUser method to get the user's confirmation status.
* 3. Invokes the ResendConfirmationCode method if the user requested another code.
* 4. Invokes the confirmSignUp method.
* 5. Invokes the AdminInitiateAuth to sign in. This results in being prompted to set up TOTP (time-based one-time password). (The response is "ChallengeName": "MFA_SETUP").
* 6. Invokes the AssociateSoftwareToken method to generate a TOTP MFA private key. This can be used with Google Authenticator.
* 7. Invokes the VerifySoftwareToken method to verify the TOTP and register for MFA.
* 8. Invokes the AdminInitiateAuth to sign in again. This results in being prompted to submit a TOTP (Response: "ChallengeName": "SOFTWARE_TOKEN_MFA").
* 9. Invokes the AdminRespondToAuthChallenge to get back a token.
*/
public class CognitoMVP {
    public static final String DASHES = new String(new char[80]).replace("\0", "-");

    public static void main(String[] args) throws NoSuchAlgorithmException,
    InvalidKeyException {
        final String usage = """

            Usage:
            <clientId> <poolId>

            Where:
            clientId - The app client Id value that you can get from the
            AWS CDK script.
            poolId - The pool Id that you can get from the AWS CDK
            script.\s
            """;
```

```
if (args.length != 2) {
    System.out.println(usage);
    System.exit(1);
}

String clientId = args[0];
String poolId = args[1];
CognitoIdentityProviderClient identityProviderClient =
CognitoIdentityProviderClient.builder()
    .region(Region.US_EAST_1)
    .build();

System.out.println(DASHES);
System.out.println("Welcome to the Amazon Cognito example scenario.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("*** Enter your user name");
Scanner in = new Scanner(System.in);
String userName = in.nextLine();

System.out.println("*** Enter your password");
String password = in.nextLine();

System.out.println("*** Enter your email");
String email = in.nextLine();

System.out.println("1. Signing up " + userName);
signUp(identityProviderClient, clientId, userName, password, email);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("2. Getting " + userName + " in the user pool");
getAdminUser(identityProviderClient, userName, poolId);

System.out
    .println("*** Conformation code sent to " + userName + ". Would
you like to send a new code? (Yes/No)");
System.out.println(DASHES);

System.out.println(DASHES);
String ans = in.nextLine();

if (ans.compareTo("Yes") == 0) {
```

```
        resendConfirmationCode(identityProviderClient, clientId, userName);
        System.out.println("3. Sending a new confirmation code");
    }
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("4. Enter confirmation code that was emailed");
    String code = in.nextLine();
    confirmSignUp(identityProviderClient, clientId, code, userName);
    System.out.println("Rechecking the status of " + userName + " in the user
pool");
    getAdminUser(identityProviderClient, userName, poolId);
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("5. Invokes the initiateAuth to sign in");
    AdminInitiateAuthResponse authResponse =
initiateAuth(identityProviderClient, clientId, userName, password,
            poolId);
    String mySession = authResponse.session();
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("6. Invokes the AssociateSoftwareToken method to
generate a TOTP key");
    String newSession = getSecretForAppMFA(identityProviderClient,
mySession);
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("*** Enter the 6-digit code displayed in Google
Authenticator");
    String myCode = in.nextLine();
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("7. Verify the TOTP and register for MFA");
    verifyTOTP(identityProviderClient, newSession, myCode);
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("8. Re-enter a 6-digit code displayed in Google
Authenticator");
    String mfaCode = in.nextLine();
```

```
        AdminInitiateAuthResponse authResponse1 =
initiateAuth(identityProviderClient, clientId, userName, password,
            poolId);
System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Invokes the AdminRespondToAuthChallenge");
        String session2 = authResponse1.session();
        adminRespondToAuthChallenge(identityProviderClient, userName, clientId,
mfaCode, session2);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("All Amazon Cognito operations were successfully
performed");
        System.out.println(DASHES);
    }

    // Respond to an authentication challenge.
    public static void adminRespondToAuthChallenge(CognitoIdentityProviderClient
identityProviderClient,
                                                String userName, String clientId, String mfaCode, String session) {
    System.out.println("SOFTWARE_TOKEN_MFA challenge is generated");
    Map<String, String> challengeResponses = new HashMap<>();

    challengeResponses.put("USERNAME", userName);
    challengeResponses.put("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

    AdminRespondToAuthChallengeRequest respondToAuthChallengeRequest =
AdminRespondToAuthChallengeRequest.builder()
        .challengeName(ChallengeNameType.SOFTWARE_TOKEN_MFA)
        .clientId(clientId)
        .challengeResponses(challengeResponses)
        .session(session)
        .build();

    AdminRespondToAuthChallengeResponse respondToAuthChallengeResult =
identityProviderClient
        .adminRespondToAuthChallenge(respondToAuthChallengeRequest);

    System.out.println("respondToAuthChallengeResult.getAuthenticationResult()"
+ respondToAuthChallengeResult.authenticationResult());
}
```

```
// Verify the TOTP and register for MFA.
public static void verifyTOTP(CognitoIdentityProviderClient
identityProviderClient, String session, String code) {
    try {
        VerifySoftwareTokenRequest tokenRequest =
VerifySoftwareTokenRequest.builder()
            .userCode(code)
            .session(session)
            .build();

        VerifySoftwareTokenResponse verifyResponse =
identityProviderClient.verifySoftwareToken(tokenRequest);
        System.out.println("The status of the token is " +
verifyResponse.statusAsString());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static AdminInitiateAuthResponse
initiateAuth(CognitoIdentityProviderClient identityProviderClient,
              String clientId, String userName, String password, String userPoolId)
{
    try {
        Map<String, String> authParameters = new HashMap<>();
        authParameters.put("USERNAME", userName);
        authParameters.put("PASSWORD", password);

        AdminInitiateAuthRequest authRequest =
AdminInitiateAuthRequest.builder()
            .clientId(clientId)
            .userPoolId(userPoolId)
            .authParameters(authParameters)
            .authFlow(AuthFlowType.ADMIN_USER_PASSWORD_AUTH)
            .build();

        AdminInitiateAuthResponse response =
identityProviderClient.adminInitiateAuth(authRequest);
        System.out.println("Result Challenge is : " +
response.challengeName());
        return response;
    }
}
```

```
        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }

    public static String getSecretForAppMFA(CognitoIdentityProviderClient identityProviderClient, String session) {
        AssociateSoftwareTokenRequest softwareTokenRequest =
        AssociateSoftwareTokenRequest.builder()
            .session(session)
            .build();

        AssociateSoftwareTokenResponse tokenResponse = identityProviderClient
            .associateSoftwareToken(softwareTokenRequest);
        String secretCode = tokenResponse.secretCode();
        System.out.println("Enter this token into Google Authenticator");
        System.out.println(secretCode);
        return tokenResponse.session();
    }

    public static void confirmSignUp(CognitoIdentityProviderClient identityProviderClient, String clientId, String code,
        String userName) {
        try {
            ConfirmSignUpRequest signUpRequest = ConfirmSignUpRequest.builder()
                .clientId(clientId)
                .confirmationCode(code)
                .username(userName)
                .build();

            identityProviderClient.confirmSignUp(signUpRequest);
            System.out.println(userName + " was confirmed");

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void resendConfirmationCode(CognitoIdentityProviderClient identityProviderClient, String clientId,
```

```
        String userName) {
    try {
        ResendConfirmationCodeRequest codeRequest =
ResendConfirmationCodeRequest.builder()
            .clientId(clientId)
            .username(userName)
            .build();

        ResendConfirmationCodeResponse response =
identityProviderClient.resendConfirmationCode(codeRequest);
        System.out.println("Method of delivery is " +
response.codeDeliveryDetails().deliveryMediumAsString());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void signUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String userName,
String password, String email) {
AttributeType userAttrs = AttributeType.builder()
    .name("email")
    .value(email)
    .build();

List<AttributeType> userAttrsList = new ArrayList<>();
userAttrsList.add(userAttrs);
try {
    SignUpRequest signUpRequest = SignUpRequest.builder()
        .userAttributes(userAttrsList)
        .username(userName)
        .clientId(clientId)
        .password(password)
        .build();

    identityProviderClient.signUp(signUpRequest);
    System.out.println("User has been signed up ");

} catch (CognitoIdentityProviderException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
```

```
}

    public static void getAdminUser(CognitoIdentityProviderClient
identityProviderClient, String userName,
        String poolId) {
    try {
        Admin GetUserRequest userRequest = Admin GetUserRequest.builder()
            .username(userName)
            .userPoolId(poolId)
            .build();

        Admin GetUserResponse response =
identityProviderClient.admin GetUser(userRequest);
        System.out.println("User status " + response.userStatusAsString());

    } catch (CognitoIdentityProviderException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
 - [Admin GetUser](#)
 - [Admin Initiate Auth](#)
 - [Admin Respond To Auth Challenge](#)
 - [Associate Software Token](#)
 - [Confirm Device](#)
 - [Confirm Sign Up](#)
 - [Initiate Auth](#)
 - [List Users](#)
 - [Resend Confirmation Code](#)
 - [Respond To Auth Challenge](#)
 - [Sign Up](#)
 - [Verify Software Token](#)

JavaScript

SDK for JavaScript (v3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

For the best experience, clone the GitHub repository and run this example. The following code represents a sample of the full example application.

```
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-
up' command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * Add logic here to handle the sign-up command
     */
  } catch (error) {
    logger.error(`Error during sign-up: ${error.message}`);
  }
};
```

```
* @type {string[]}
*/
const values = getSecondValuesFromEntries(FILE_USER_POOLS);
const clientId = values[0];
validateClient(clientId);
logger.log("Signing up.");
await signUp({ clientId, username, password, email });
logger.log(`Signed up. A confirmation email has been sent to: ${email}.`);
logger.log(
  `Run 'confirm-sign-up ${username} <code>' to confirm your account.`,
);
} catch (err) {
  logger.error(err);
}
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};
```

```
const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the
      'confirm-sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
      'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Confirming user.");
    await confirmSignUp({ clientId, username, code });
    logger.log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
      in.`,
    );
  } catch (err) {
    logger.error(err);
  }
};

export { confirmSignUpHandler };
```

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../../../actions/associate-software-token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-software-token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
```

```
        throw new Error(
          `User pool client id is missing. Did you run 'create-user-pool'?`,
        );
      }
    };

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-
initiate-auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    logger.log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      logger.log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }
  }
};
```

```
if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
    handleSoftwareTokenMfa(Session);
    logger.log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
}
} catch (err) {
    logger.error(err);
}
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new AdminInitiateAuthCommand({
        ClientId: clientId,
        UserPoolId: userPoolId,
        AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        AuthParameters: { USERNAME: username, PASSWORD: password },
    });

    return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../../../actions/admin-respond-to-auth-challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
    if (!username) {
        throw new Error(
            `Username is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`,
        );
    }
};

const verifyTotp = (totp) => {
    if (!totp) {
        throw new Error(
            `Time-based one-time password (TOTP) is missing. It must be provided as an argument to the 'admin-respond-to-auth-challenge' command.`,
        );
    }
};
```

```
        );
    }
};

const storeAccessToken = (token) => {
    process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
    const [_, username, totp] = commands;

    try {
        verifyUsername(username);
        verifyTotp(totp);

        const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
        const session = process.env.SESSION;

        const { AuthenticationResult } = await adminRespondToAuthChallenge({
            clientId,
            userPoolId,
            username,
            totp,
            session,
        });

        storeAccessToken(AuthenticationResult.AccessToken);

        logger.log("Successfully authenticated.");
    } catch (err) {
        logger.error(err);
    }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
    clientId,
    username,
    session,
    userPoolId,
    code,
}) => {
    const client = new CognitoIdentityProviderClient({});
```

```
const command = new RespondToAuthChallengeCommand({
  ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
  ChallengeResponses: {
    SOFTWARE_TOKEN_MFA_CODE: code,
    USERNAME: username,
  },
  ClientId: clientId,
  UserPoolId: userPoolId,
  Session: session,
});

return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-software-token' command.`,
    );
  }
};

const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    logger.log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    logger.log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    logger.error(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});
```

```
// The 'Session' is provided in the response to 'AssociateSoftwareToken'.
const session = process.env.SESSION;

if (!session) {
    throw new Error(
        "Missing a valid Session. Did you run 'admin-initiate-auth?'",
    );
}

const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
});

return client.send(command);
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
 - [Admin GetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
/**
```

Before running this Kotlin code example, set up your development environment, including your credentials.

For more information, see the following documentation:

<https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html>

TIP: To set up the required user pool, run the AWS Cloud Development Kit (AWS CDK) script provided in this GitHub repo at `resources/cdk/cognito_scenario_user_pool_with_mfa`.

This code example performs the following operations:

1. Invokes the `signUp` method to sign up a user.
 2. Invokes the `admin GetUser` method to get the user's confirmation status.
 3. Invokes the `ResendConfirmationCode` method if the user requested another code.
 4. Invokes the `confirmSignUp` method.
 5. Invokes the `initiateAuth` to sign in. This results in being prompted to set up TOTP (time-based one-time password). (The response is "ChallengeName": "MFA_SETUP").
 6. Invokes the `AssociateSoftwareToken` method to generate a TOTP MFA private key. This can be used with Google Authenticator.
 7. Invokes the `VerifySoftwareToken` method to verify the TOTP and register for MFA.
 8. Invokes the `AdminInitiateAuth` to sign in again. This results in being prompted to submit a TOTP (Response: "ChallengeName": "SOFTWARE_TOKEN_MFA").
 9. Invokes the `AdminRespondToAuthChallenge` to get back a token.
- */

```
suspend fun main(args: Array<String>) {  
    val usage = """
```

```
Usage:  
    <clientId> <poolId>  
Where:  
    clientId - The app client Id value that you can get from the AWS CDK  
script.  
    poolId - The pool Id that you can get from the AWS CDK script.  
....  
  
if (args.size != 2) {  
    println(usage)  
    exitProcess(1)  
}  
  
val clientId = args[0]  
val poolId = args[1]  
  
// Use the console to get data from the user.  
println("/** Enter your use name")  
val in0b = Scanner(System.`in`)  
val userName = in0b.nextLine()  
println(userName)  
  
println("/** Enter your password")  
val password: String = in0b.nextLine()  
  
println("/** Enter your email")  
val email = in0b.nextLine()  
  
println("/** Signing up $userName")  
signUp(clientId, userName, password, email)  
  
println("/** Getting $userName in the user pool")  
getAdminUser(userName, poolId)  
  
println("/** Conformation code sent to $userName. Would you like to send a  
new code? (Yes/No)")  
val ans = in0b.nextLine()  
  
if (ans.compareTo("Yes") == 0) {  
    println("/** Sending a new confirmation code")  
    resendConfirmationCode(clientId, userName)  
}  
println("/** Enter the confirmation code that was emailed")  
val code = in0b.nextLine()
```

```
confirmSignUp(clientId, code, userName)

println("/** Rechecking the status of $userName in the user pool")
getAdminUser(userName, poolId)

val authResponse = checkAuthMethod(clientId, userName, password, poolId)
val mySession = authResponse.session
val newSession = getSecretForAppMFA(mySession)
println("/** Enter the 6-digit code displayed in Google Authenticator")
val myCode = inOb.nextLine()

// Verify the TOTP and register for MFA.
verifyTOTP(newSession, myCode)
println("/** Re-enter a 6-digit code displayed in Google Authenticator")
val mfaCode: String = inOb.nextLine()
val authResponse1 = checkAuthMethod(clientId, userName, password, poolId)
val session2 = authResponse1.session
adminRespondToAuthChallenge(userName, clientId, mfaCode, session2)
}

suspend fun checkAuthMethod(
    clientIdVal: String,
    userNameVal: String,
    passwordVal: String,
    userPoolIdVal: String,
): AdminInitiateAuthResponse {
    val authParas = mutableMapOf<String, String>()
    authParas["USERNAME"] = userNameVal
    authParas["PASSWORD"] = passwordVal

    val authRequest =
        AdminInitiateAuthRequest {
            clientId = clientIdVal
            userPoolId = userPoolIdVal
            authParameters = authParas
            authFlow = AuthFlowType.AdminUserPasswordAuth
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val response = identityProviderClient.adminInitiateAuth(authRequest)
        println("Result Challenge is ${response.challengeName}")
        return response
    }
}
```

```
}

suspend fun resendConfirmationCode(
    clientIdVal: String?,
    userNameVal: String?,
) {
    val codeRequest =
        ResendConfirmationCodeRequest {
            clientId = clientIdVal
            username = userNameVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val response = identityProviderClient.resendConfirmationCode(codeRequest)
        println("Method of delivery is " +
        (response.codeDeliveryDetails?.deliveryMedium))
    }
}

// Respond to an authentication challenge.
suspend fun adminRespondToAuthChallenge(
    userName: String,
    clientIdVal: String?,
    mfaCode: String,
    sessionVal: String?,
) {
    println("SOFTWARE_TOKEN_MFA challenge is generated")
    val challengeResponses0b = mutableMapOf<String, String>()
    challengeResponses0b["USERNAME"] = userName
    challengeResponses0b["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

    val adminRespondToAuthChallengeRequest =
        AdminRespondToAuthChallengeRequest {
            challengeName = ChallengeNameType.SoftwareTokenMfa
            clientId = clientIdVal
            challengeResponses = challengeResponses0b
            session = sessionVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val respondToAuthChallengeResult =
        identityProviderClient.adminRespondToAuthChallenge(adminRespondToAuthChallengeRequest)
```

```
        println("respondToAuthChallengeResult.getAuthenticationResult()
${respondToAuthChallengeResult.authenticationResult}")
    }
}

// Verify the TOTP and register for MFA.
suspend fun verifyTOTP(
    sessionVal: String?,
    codeVal: String?,
) {
    val tokenRequest =
        VerifySoftwareTokenRequest {
            userCode = codeVal
            session = sessionVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val verifyResponse =
            identityProviderClient.verifySoftwareToken(tokenRequest)
        println("The status of the token is ${verifyResponse.status}")
    }
}

suspend fun getSecretForAppMFA(sessionVal: String?): String? {
    val softwareTokenRequest =
        AssociateSoftwareTokenRequest {
            session = sessionVal
        }

    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        val tokenResponse =
            identityProviderClient.associateSoftwareToken(softwareTokenRequest)
        val secretCode = tokenResponse.secretCode
        println("Enter this token into Google Authenticator")
        println(secretCode)
        return tokenResponse.session
    }
}

suspend fun confirmSignUp(
    clientIdVal: String?,
    codeVal: String?,
```

```
        userNameVal: String?,  
    ) {  
        val signUpRequest =  
            ConfirmSignUpRequest {  
                clientId = clientIdVal  
                confirmationCode = codeVal  
                username = userNameVal  
            }  
  
        CognitoIdentityProviderClient { region = "us-east-1" }.use  
        { identityProviderClient ->  
            identityProviderClient.confirmSignUp(signUpRequest)  
            println("$userNameVal was confirmed")  
        }  
    }  
  
    suspend fun getAdminUser(  
        userNameVal: String?,  
        poolIdVal: String?,  
    ) {  
        val userRequest =  
            Admin GetUserRequest {  
                username = userNameVal  
                userPoolId = poolIdVal  
            }  
  
        CognitoIdentityProviderClient { region = "us-east-1" }.use  
        { identityProviderClient ->  
            val response = identityProviderClient.admin GetUser(userRequest)  
            println("User status ${response.userStatus}")  
        }  
    }  
  
    suspend fun signUp(  
        clientIdVal: String?,  
        userNameVal: String?,  
        passwordVal: String?,  
        emailVal: String?,  
    ) {  
        val userAttrs =  
            AttributeType {  
                name = "email"  
                value = emailVal  
            }  
    }
```

```
    val userAttrsList = mutableListOf<AttributeType>()
    userAttrsList.add(userAttrs)
    val signUpRequest =
        SignUpRequest {
            userAttributes = userAttrsList
            username = userNameVal
            clientId = clientIdVal
            password = passwordVal
        }
    CognitoIdentityProviderClient { region = "us-east-1" }.use
    { identityProviderClient ->
        identityProviderClient.signUp(signUpRequest)
        println("User has been signed up")
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
 - [Admin GetUser](#)
 - [Admin Initiate Auth](#)
 - [Admin Respond To Auth Challenge](#)
 - [Associate Software Token](#)
 - [Confirm Device](#)
 - [Confirm Sign Up](#)
 - [Initiate Auth](#)
 - [List Users](#)
 - [Resend Confirmation Code](#)
 - [Respond To Auth Challenge](#)
 - [Sign Up](#)
 - [Verify Software Token](#)

Python

SDK for Python (Boto3)

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Create a class that wraps Amazon Cognito functions used in the scenario.

```
class CognitoIdentityProviderWrapper:  
    """Encapsulates Amazon Cognito actions"""  
  
    def __init__(self, cognito_idp_client, user_pool_id, client_id,  
                 client_secret=None):  
        """  
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider  
        client.  
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.  
        :param client_id: The ID of a client application registered with the user  
        pool.  
        :param client_secret: The client secret, if the client has a secret.  
        """  
        self.cognito_idp_client = cognito_idp_client  
        self.user_pool_id = user_pool_id  
        self.client_id = client_id  
        self.client_secret = client_secret  
  
    def _secret_hash(self, user_name):  
        """  
        Calculates a secret hash from a user name and a client secret.  
        :param user_name: The user name to use when calculating the hash.  
        :return: The secret hash.  
        """  
        key = self.client_secret.encode()  
        msg = bytes(user_name + self.client_id, "utf-8")  
        secret_hash = base64.b64encode(  
            hmac.new(key, msg, digestmod=hashlib.sha256).digest()  
        ).decode()
```

```
logger.info("Made secret hash for %s: %s.", user_name, secret_hash)
return secret_hash

def sign_up_user(self, user_name, password, user_email):
    """
    Signs up a new user with Amazon Cognito. This action prompts Amazon
    Cognito
        to send an email to the specified email address. The email contains a
    code that
        can be used to confirm the user.

    When the user already exists, the user status is checked to determine
    whether
        the user has been confirmed.

    :param user_name: The user name that identifies the new user.
    :param password: The password for the new user.
    :param user_email: The email address for the new user.
    :return: True when the user is already confirmed with Amazon Cognito.
        Otherwise, false.
    """
    try:
        kwargs = {
            "ClientId": self.client_id,
            "Username": user_name,
            "Password": password,
            "UserAttributes": [{"Name": "email", "Value": user_email}],
        }
        if self.client_secret is not None:
            kwargs["SecretHash"] = self._secret_hash(user_name)
        response = self.cognito_idp_client.sign_up(**kwargs)
        confirmed = response["UserConfirmed"]
    except ClientError as err:
        if err.response["Error"]["Code"] == "UsernameExistsException":
            response = self.cognito_idp_client.admin_get_user(
                UserPoolId=self.user_pool_id, Username=user_name
            )
            logger.warning(
                "User %s exists and is %s.", user_name,
            response["UserStatus"]
            )
            confirmed = response["UserStatus"] == "CONFIRMED"
        else:
            logger.error(
```

```
        "Couldn't sign up %s. Here's why: %s: %s",
        user_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
return confirmed


def resend_confirmation(self, user_name):
    """
    Prompts Amazon Cognito to resend an email with a new confirmation code.

    :param user_name: The name of the user who will receive the email.
    :return: Delivery information about where the email is sent.
    """
    try:
        kwargs = {"ClientId": self.client_id, "Username": user_name}
        if self.client_secret is not None:
            kwargs["SecretHash"] = self._secret_hash(user_name)
        response = self.cognito_idp_client.resend_confirmation_code(**kwargs)
        delivery = response["CodeDeliveryDetails"]
    except ClientError as err:
        logger.error(
            "Couldn't resend confirmation to %s. Here's why: %s: %s",
            user_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return delivery


def confirm_user_sign_up(self, user_name, confirmation_code):
    """
    Confirms a previously created user. A user must be confirmed before they
    can sign in to Amazon Cognito.

    :param user_name: The name of the user to confirm.
    :param confirmation_code: The confirmation code sent to the user's
    registered
                           email address.
    :return: True when the confirmation succeeds.
    """
```

```
"""
try:
    kwargs = {
        "ClientId": self.client_id,
        "Username": user_name,
        "ConfirmationCode": confirmation_code,
    }
    if self.client_secret is not None:
        kwargs["SecretHash"] = self._secret_hash(user_name)
    self.cognito_idp_client.confirm_sign_up(**kwargs)
except ClientError as err:
    logger.error(
        "Couldn't confirm sign up for %s. Here's why: %s: %s",
        user_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return True

def list_users(self):
    """
    Returns a list of the users in the current user pool.

    :return: The list of users.
    """
    try:
        response =
self.cognito_idp_client.list_users(UserPoolId=self.user_pool_id)
        users = response["Users"]
    except ClientError as err:
        logger.error(
            "Couldn't list users for %s. Here's why: %s: %s",
            self.user_pool_id,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return users
```

```
def start_sign_in(self, user_name, password):
    """
    Starts the sign-in process for a user by using administrator credentials.
    This method of signing in is appropriate for code running on a secure
    server.

    If the user pool is configured to require MFA and this is the first sign-
    in
        for the user, Amazon Cognito returns a challenge response to set up an
        MFA application. When this occurs, this function gets an MFA secret from
        Amazon Cognito and returns it to the caller.

    :param user_name: The name of the user to sign in.
    :param password: The user's password.
    :return: The result of the sign-in attempt. When sign-in is successful,
    this
            returns an access token that can be used to get AWS credentials.
    Otherwise,
        Amazon Cognito returns a challenge to set up an MFA application,
        or a challenge to enter an MFA code from a registered MFA
        application.
    """
    try:
        kwargs = {
            "UserPoolId": self.user_pool_id,
            "ClientId": self.client_id,
            "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",
            "AuthParameters": {"USERNAME": user_name, "PASSWORD": password},
        }
        if self.client_secret is not None:
            kwargs["AuthParameters"]["SECRET_HASH"] =
self._secret_hash(user_name)
            response = self.cognito_idp_client.admin_initiate_auth(**kwargs)
            challenge_name = response.get("ChallengeName", None)
            if challenge_name == "MFA_SETUP":
                if (
                    "SOFTWARE_TOKEN_MFA"
                    in response["ChallengeParameters"]["MFAS_CAN_SETUP"]
                ):
                    response.update(self.get_mfa_secret(response["Session"]))
            else:
                raise RuntimeError(
                    "The user pool requires MFA setup, but the user pool is
not "
```

```
        "configured for TOTP MFA. This example requires TOTP
MFA."
    )
except ClientError as err:
    logger.error(
        "Couldn't start sign in for %s. Here's why: %s: %s",
        user_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    response.pop("ResponseMetadata", None)
    return response

def get_mfa_secret(self, session):
    """
    Gets a token that can be used to associate an MFA application with the
    user.

    :param session: Session information returned from a previous call to
    initiate
                    authentication.
    :return: An MFA token that can be used to set up an MFA application.
    """
    try:
        response =
self.cognito_idp_client.associate_software_token(Session=session)
    except ClientError as err:
        logger.error(
            "Couldn't get MFA secret. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        response.pop("ResponseMetadata", None)
        return response

def verify_mfa(self, session, user_code):
    """
    Verify a new MFA application that is associated with a user.

```

```
:param session: Session information returned from a previous call to
initiate
    authentication.
:param user_code: A code generated by the associated MFA application.
:return: Status that indicates whether the MFA application is verified.
"""
try:
    response = self.cognito_idp_client.verify_software_token(
        Session=session, UserCode=user_code
    )
except ClientError as err:
    logger.error(
        "Couldn't verify MFA. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    response.pop("ResponseMetadata", None)
    return response

def respond_to_mfa_challenge(self, user_name, session, mfa_code):
    """
    Responds to a challenge for an MFA code. This completes the second step
    of
    a two-factor sign-in. When sign-in is successful, it returns an access
    token
    that can be used to get AWS credentials from Amazon Cognito.

    :param user_name: The name of the user who is signing in.
    :param session: Session information returned from a previous call to
    initiate
        authentication.
    :param mfa_code: A code generated by the associated MFA application.
    :return: The result of the authentication. When successful, this contains
    an
        access token for the user.
    """
try:
    kwargs = {
        "UserPoolId": self.user_pool_id,
        "ClientId": self.client_id,
```

```
        "ChallengeName": "SOFTWARE_TOKEN_MFA",
        "Session": session,
        "ChallengeResponses": {
            "USERNAME": user_name,
            "SOFTWARE_TOKEN_MFA_CODE": mfa_code,
        },
    }
    if self.client_secret is not None:
        kwargs["ChallengeResponses"]["SECRET_HASH"] = self._secret_hash(
            user_name
        )
    response =
self.cognito_idp_client.admin_respond_to_auth_challenge(**kwargs)
    auth_result = response["AuthenticationResult"]
except ClientError as err:
    if err.response["Error"]["Code"] == "ExpiredCodeException":
        logger.warning(
            "Your MFA code has expired or has been used already. You
might have "
            "to wait a few seconds until your app shows you a new code."
        )
    else:
        logger.error(
            "Couldn't respond to mfa challenge for %s. Here's why: %s:
%s",
            user_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return auth_result

def confirm_mfa_device(
    self,
    user_name,
    device_key,
    device_group_key,
    device_password,
    access_token,
    aws_srp,
):
    """
```

Confirms an MFA device to be tracked by Amazon Cognito. When a device is tracked, its key and password can be used to sign in without requiring a new MFA code from the MFA application.

:param user_name: The user that is associated with the device.
:param device_key: The key of the device, returned by Amazon Cognito.
:param device_group_key: The group key of the device, returned by Amazon Cognito.
:param device_password: The password that is associated with the device.
:param access_token: The user's access token.
:param aws_srp: A class that helps with Secure Remote Password (SRP) calculations. The scenario associated with this example uses the warrant package.

:return: True when the user must confirm the device. Otherwise, False.

When False, the device is automatically confirmed and tracked.

```
"""
srp_helper = aws_srp.AWSSRP(
    username=user_name,
    password=device_password,
    pool_id="_",
    client_id=self.client_id,
    client_secret=None,
    client=self.cognito_idp_client,
)
device_and_pw = f"{device_group_key}{device_key}:{device_password}"
device_and_pw_hash = aws_srp.hash_sha256(device_and_pw.encode("utf-8"))
salt = aws_srp.pad_hex(aws_srp.get_random(16))
x_value = aws_srp.hex_to_long(aws_srp.hex_hash(salt +
device_and_pw_hash))
verifier = aws_srp.pad_hex(pow(srp_helper.val_g, x_value,
srp_helper.big_n))
device_secret_verifier_config = {
    "PasswordVerifier": base64.standard_b64encode(
        bytearray.fromhex(verifier)
    ).decode("utf-8"),
    "Salt": base64.standard_b64encode(bytearray.fromhex(salt)).decode("utf-8"),
}
try:
    response = self.cognito_idp_client.confirm_device(
        AccessToken=access_token,
```

```
        DeviceKey=device_key,
        DeviceSecretVerifierConfig=device_secret_verifier_config,
    )
    user_confirm = response["UserConfirmationNecessary"]
except ClientError as err:
    logger.error(
        "Couldn't confirm mfa device %s. Here's why: %s: %s",
        device_key,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return user_confirm

def sign_in_with_tracked_device(
    self,
    user_name,
    password,
    device_key,
    device_group_key,
    device_password,
    aws_srp,
):
    """
    Signs in to Amazon Cognito as a user who has a tracked device. Signing in
    with a tracked device lets a user sign in without entering a new MFA
    code.

    Signing in with a tracked device requires that the client respond to the
    SRP protocol. The scenario associated with this example uses the warrant
    package
    to help with SRP calculations.

    For more information on SRP, see https://en.wikipedia.org/wiki/Secure\_Remote\_Password\_protocol.
    :param user_name: The user that is associated with the device.
    :param password: The user's password.
    :param device_key: The key of a tracked device.
    :param device_group_key: The group key of a tracked device.
    :param device_password: The password that is associated with the device.

```

```
:param aws_srp: A class that helps with SRP calculations. The scenario
                associated with this example uses the warrant package.
:return: The result of the authentication. When successful, this contains
an
        access token for the user.

"""
try:
    srp_helper = aws_srp.AWSSRP(
        username=user_name,
        password=device_password,
        pool_id="_",
        client_id=self.client_id,
        client_secret=None,
        client=self.cognito_idp_client,
    )

    response_init = self.cognito_idp_client.initiate_auth(
        ClientId=self.client_id,
        AuthFlow="USER_PASSWORD_AUTH",
        AuthParameters={
            "USERNAME": user_name,
            "PASSWORD": password,
            "DEVICE_KEY": device_key,
        },
    )
    if response_init["ChallengeName"] != "DEVICE_SRP_AUTH":
        raise RuntimeError(
            f"Expected DEVICE_SRP_AUTH challenge but got "
            f"{response_init['ChallengeName']}."
        )

    auth_params = srp_helper.get_auth_params()
    auth_params["DEVICE_KEY"] = device_key
    response_auth = self.cognito_idp_client.respond_to_auth_challenge(
        ClientId=self.client_id,
        ChallengeName="DEVICE_SRP_AUTH",
        ChallengeResponses=auth_params,
    )
    if response_auth["ChallengeName"] != "DEVICE_PASSWORD_VERIFIER":
        raise RuntimeError(
            f"Expected DEVICE_PASSWORD_VERIFIER challenge but got "
            f"{response_init['ChallengeName']}."
        )


```

```
challenge_params = response_auth["ChallengeParameters"]
challenge_params["USER_ID_FOR_SRP"] = device_group_key + device_key
cr = srp_helper.process_challenge(challenge_params, {"USERNAME": user_name})
cr["USERNAME"] = user_name
cr["DEVICE_KEY"] = device_key
response_verifier =
self.cognito_idp_client.respond_to_auth_challenge(
    ClientId=self.client_id,
    ChallengeName="DEVICE_PASSWORD_VERIFIER",
    ChallengeResponses=cr,
)
auth_tokens = response_verifier["AuthenticationResult"]
except ClientError as err:
    logger.error(
        "Couldn't start client sign in for %s. Here's why: %s: %s",
        user_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return auth_tokens
```

Create a class that runs the scenario. This example also registers an MFA device to be tracked by Amazon Cognito and shows you how to sign in by using a password and information from the tracked device. This avoids the need to enter a new MFA code.

```
def run_scenario(cognito_idp_client, user_pool_id, client_id):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon Cognito user signup with MFA demo.")
    print("-" * 88)

    cog_wrapper = CognitoIdentityProviderWrapper(
        cognito_idp_client, user_pool_id, client_id
    )
```

```
user_name = q.ask("Let's sign up a new user. Enter a user name: ",  
q.non_empty)  
password = q.ask("Enter a password for the user: ", q.non_empty)  
email = q.ask("Enter a valid email address that you own: ", q.non_empty)  
confirmed = cog_wrapper.sign_up_user(user_name, password, email)  
while not confirmed:  
    print(  
        f"User {user_name} requires confirmation. Check {email} for "  
        f"a verification code."  
    )  
    confirmation_code = q.ask("Enter the confirmation code from the email: ")  
    if not confirmation_code:  
        if q.ask("Do you need another confirmation code (y/n)? ",  
q.is_yesno):  
            delivery = cog_wrapper.resend_confirmation(user_name)  
            print(  
                f"Confirmation code sent by {delivery['DeliveryMedium']} "  
                f"to {delivery['Destination']}."  
            )  
        else:  
            confirmed = cog_wrapper.confirm_user_sign_up(user_name,  
confirmation_code)  
            print(f"User {user_name} is confirmed and ready to use.")  
            print("-" * 88)  
  
    print("Let's get a list of users in the user pool.")  
    q.ask("Press Enter when you're ready.")  
    users = cog_wrapper.list_users()  
    if users:  
        print(f"Found {len(users)} users:")  
        pp(users)  
    else:  
        print("No users found.")  
    print("-" * 88)  
  
    print("Let's sign in and get an access token.")  
    auth_tokens = None  
    challenge = "ADMIN_USER_PASSWORD_AUTH"  
    response = {}  
    while challenge is not None:  
        if challenge == "ADMIN_USER_PASSWORD_AUTH":  
            response = cog_wrapper.start_sign_in(user_name, password)  
            challenge = response["ChallengeName"]  
        elif response["ChallengeName"] == "MFA_SETUP":
```

```
        print("First, we need to set up an MFA application.")
        qr_img = qrcode.make(
            f"otpauth://totp/{user_name}?secret={response['SecretCode']}"
        )
        qr_img.save("qr.png")
        q.ask(
            "Press Enter to see a QR code on your screen. Scan it into an MFA"
        )
        "application, such as Google Authenticator."
    )
    webbrowser.open("qr.png")
    mfa_code = q.ask(
        "Enter the verification code from your MFA application: ",
        q.non_empty
    )
    response = cog_wrapper.verify_mfa(response["Session"], mfa_code)
    print(f"MFA device setup {response['Status']}"))
    print("Now that an MFA application is set up, let's sign in again.")
    print(
        "You might have to wait a few seconds for a new MFA code to"
        "appear in "
    )
    "your MFA application."
)
challenge = "ADMIN_USER_PASSWORD_AUTH"
elif response["ChallengeName"] == "SOFTWARE_TOKEN_MFA":
    auth_tokens = None
    while auth_tokens is None:
        mfa_code = q.ask(
            "Enter a verification code from your MFA application: ",
            q.non_empty
        )
        auth_tokens = cog_wrapper.respond_to_mfa_challenge(
            user_name, response["Session"], mfa_code
        )
    print(f"You're signed in as {user_name}.")
    print("Here's your access token:")
    pp(auth_tokens["AccessToken"])
    print("And your device information:")
    pp(auth_tokens["NewDeviceMetadata"])
    challenge = None
else:
    raise Exception(f"Got unexpected challenge {response['ChallengeName']}"))
    print("-" * 88)
```

```
device_group_key = auth_tokens["NewDeviceMetadata"]["DeviceGroupKey"]
device_key = auth_tokens["NewDeviceMetadata"]["DeviceKey"]
device_password = base64.standard_b64encode(os.urandom(40)).decode("utf-8")

print("Let's confirm your MFA device so you don't have re-enter MFA tokens
for it.")
q.ask("Press Enter when you're ready.")
cog_wrapper.confirm_mfa_device(
    user_name,
    device_key,
    device_group_key,
    device_password,
    auth_tokens["AccessToken"],
    aws_srp,
)
print(f"Your device {device_key} is confirmed.")
print("-" * 88)

print(
    f"Now let's sign in as {user_name} from your confirmed device
{device_key}.\n"
    f"Because this device is tracked by Amazon Cognito, you won't have to re-
enter an MFA code."
)
q.ask("Press Enter when ready.")
auth_tokens = cog_wrapper.sign_in_with_tracked_device(
    user_name, password, device_key, device_group_key, device_password,
aws_srp
)
print("You're signed in. Your access token is:")
pp(auth_tokens["AccessToken"])
print("-" * 88)

print("Don't forget to delete your user pool when you're done with this
example.")
print("\nThanks for watching!")
print("-" * 88)

def main():
    parser = argparse.ArgumentParser(
        description="Shows how to sign up a new user with Amazon Cognito and
associate "
```

```
        "the user with an MFA application for multi-factor authentication."
    )
    parser.add_argument(
        "user_pool_id", help="The ID of the user pool to use for the example."
    )
    parser.add_argument(
        "client_id", help="The ID of the client application to use for the example."
    )
    args = parser.parse_args()
    try:
        run_scenario(boto3.client("cognito-identity"), args.user_pool_id,
args.client_id)
    except Exception:
        logging.exception("Something went wrong with the demo.")

if __name__ == "__main__":
    main()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK

The following code example shows how to write custom activity data with a Lambda function after Amazon Cognito user authentication.

- Use administrator functions to add a user to a user pool.
- Configure a user pool to call a Lambda function for the PostAuthentication trigger.
- Sign the new user in to Amazon Cognito.
- The Lambda function writes custom information to CloudWatch Logs and to an DynamoDB table.
- Get and display custom data from the DynamoDB table, then clean up resources.

Go

SDK for Go V2

 **Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
import (
    "context"
    "errors"
    "log"
    "strings"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
```

```
"github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
"github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// ActivityLog separates the steps of this scenario into individual functions so
// that
// they are simpler to read and understand.
type ActivityLog struct {
    helper      IScenarioHelper
    questioner  demotools.IQuestioner
    resources   Resources
    cognitoActor *actions.CognitoActions
}

// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner,
    helper IScenarioHelper) ActivityLog {
    scenario := ActivityLog{
        helper:      helper,
        questioner:  questioner,
        resources:   Resources{},
        cognitoActor: &actions.CognitoActions{CognitoClient:
            cognitoidentityprovider.NewFromConfig(sdkConfig)},
    }
    scenario.resources.init(scenario.cognitoActor, questioner)
    return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
// credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
    tableName string) (string, string) {
    log.Println("To facilitate this example, let's add a user to the user pool using
    administrator privileges.")
    users, err := runner.helper.GetKnownUsers(ctx, tableName)
    if err != nil {
        panic(err)
    }
    user := users.Users[0]
    log.Printf("Adding known user %v to the user pool.\n", user.UserName)
    err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
        user.UserEmail)
    if err != nil {
        panic(err)
    }
}
```

```
}

pwSet := false
password := runner.questioner.AskPassword("\nEnter a password that has at least
eight characters, uppercase, lowercase, numbers and symbols.\n"+
"(the password will not display as you type):", 8)
for !pwSet {
    log.Printf("\nSetting password for user '%v'.\n", user.UserName)
    err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
password)
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            password = runner.questioner.AskPassword("\nEnter another password:", 8)
        } else {
            panic(err)
        }
    } else {
        pwSet = true
    }
}

log.Println(strings.Repeat("-", 88))

return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
string, activityLogArn string) {
    log.Println("Let's add a Lambda function to handle the PostAuthentication
trigger from Cognito.\n" +
    "This trigger happens after a user is authenticated, and lets your function
take action, such as logging\n" +
    "the outcome.")
    err := runner.cognitoActor.UpdateTriggers(
        ctx, userPoolId,
        actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
aws.String(activityLogArn)})
    if err != nil {
        panic(err)
    }
    runner.resources.triggers = append(runner.resources.triggers,
actions.PostAuthentication)
```

```
log.Printf("Lambda function %v added to user pool %v to handle
PostAuthentication Cognito trigger.\n",
activityLogArn, userPoolId)

log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string,
userName string, password string) {
log.Printf("Now we'll sign in user %v and check the results in the logs and the
DynamoDB table.", userName)
runner.questioner.Ask("Press Enter when you're ready.")
authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
if err != nil {
panic(err)
}
log.Println("Sign in successful.",
"The PostAuthentication Lambda handler writes custom information to CloudWatch
Logs.")

runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
string, userName string) {
log.Println("The PostAuthentication handler also writes login data to the
DynamoDB table.")
runner.questioner.Ask("Press Enter when you're ready to continue.")
users, err := runner.helper.GetKnownUsers(ctx, tableName)
if err != nil {
panic(err)
}
for _, user := range users.Users {
if user.UserName == userName {
log.Println("The last login info for the user in the known users table is:")
log.Printf("\t%+v", *user.LastLogin)
}
}
log.Println(strings.Repeat("-", 88))
}
```

```
// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
    defer func() {
        if r := recover(); r != nil {
            log.Println("Something went wrong with the demo.")
            runner.resources.Cleanup(ctx)
        }
    }()
}

log.Println(strings.Repeat("-", 88))
log.Printf("Welcome\n")

log.Println(strings.Repeat("-", 88))

stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
if err != nil {
    panic(err)
}
runner.resources.userPoolId = stackOutputs["UserPoolId"]
runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
    stackOutputs["TableName"])

runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
    stackOutputs["ActivityLogFunctionArn"])
runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

runner.resources.Cleanup(ctx)

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}
```

Handle the PostAuthentication trigger with a Lambda function.

```
import (
```

```
"context"
"fmt"
"log"
"os"
"time"

"github.com/aws/aws-lambda-go/events"
"github.com/aws/aws-lambda-go/lambda"
"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/feature/dynamodb/AttributeValue"
"github.com/aws/aws-sdk-go-v2/service/dynamodb"
dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
// format.
type LoginInfo struct {
    UserPoolId string `dynamodbav:"UserPoolId"`
    ClientId   string `dynamodbav:"ClientId"`
    Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
// format.
type UserInfo struct {
    UserName  string `dynamodbav:"UserName"`
    UserEmail string `dynamodbav:"UserEmail"`
    LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
    userEmail, err := AttributeValue.Marshal(user.UserEmail)
    if err != nil {
        panic(err)
    }
    return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
    dynamoClient *dynamodb.Client
```

```
}

// HandleRequest handles the PostAuthentication event by writing custom data to
// the logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
    event events.CognitoEventUserPoolsPostAuthentication)
(events.CognitoEventUserPoolsPostAuthentication, error) {
    log.Printf("Received post authentication trigger from %v for user '%v'",
        event.TriggerSource, event.UserName)
    tableName := os.Getenv(TABLE_NAME)
    user := UserInfo{
        UserName: event.UserName,
        UserEmail: event.Request.UserAttributes["email"],
        LastLogin: LoginInfo{
            UserPoolId: event.UserPoolID,
            ClientId: event.CallerContext.ClientID,
            Time:       time.Now().Format(time.UnixDate),
        },
    }
    // Write to CloudWatch Logs.
    fmt.Printf("%#v", user)

    // Also write to an external system. This examples uses DynamoDB to demonstrate.
    userMap, err := attributevalue.MarshalMap(user)
    if err != nil {
        log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
    } else if len(userMap) == 0 {
        log.Printf("User info marshaled to an empty map.")
    } else {
        _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
            Item:      userMap,
            TableName: aws.String(tableName),
        })
        if err != nil {
            log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
        } else {
            log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
        }
    }

    return event, nil
}
```

```
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Panicln(err)
    }
    h := handler{
        dynamoClient: dynamodb.NewFromConfig(sdkConfig),
    }
    lambda.Start(h.HandleRequest)
}
```

Create a struct that performs common tasks.

```
import (
    "context"
    "log"
    "strings"
    "time"
    "user_pools_and_lambda_triggers/actions"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudformation"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
// example.
type IScenarioHelper interface {
    Pause(secs int)
    GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
        error)
    PopulateUserTable(ctx context.Context, tableName string)
    GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
    AddKnownUser(ctx context.Context, tableName string, user actions.User)
    ListRecentLogEvents(ctx context.Context, functionName string)
}
```

```
// ScenarioHelper contains AWS wrapper structs used by the workflows in this
// example.
type ScenarioHelper struct {
    questioner demotools.IQuestioner
    dynamoActor *actions.DynamoActions
    cfnActor    *actions.CloudFormationActions
    cwlActor    *actions.CloudWatchLogsActions
    isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner) ScenarioHelper {
    scenario := ScenarioHelper{
        questioner: questioner,
        dynamoActor: &actions.DynamoActions{DynamoClient:
            dynamodb.NewFromConfig(sdkConfig)},
        cfnActor:    &actions.CloudFormationActions{CfnClient:
            cloudformation.NewFromConfig(sdkConfig)},
        cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
            cloudwatchlogs.NewFromConfig(sdkConfig)},
    }
    return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
    if !helper.isTestRun {
        time.Sleep(time.Duration(secs) * time.Second)
    }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
// structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
    string) (actions.StackOutputs, error) {
    return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
    string) {
    log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
    this example.\n", tableName)
```

```
err := helper.dynamoActor.PopulateTable(ctx, tableName)
if err != nil {
    panic(err)
}
}

// GetKnownUsers gets the users from the known users table in a structured
// format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
(actions.UserList, error) {
    knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
    if err != nil {
        log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
            tableName, err)
    }
    return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
    user actions.User) {
    log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
    table...\n",
        user.UserName, user.UserEmail)
    err := helper.dynamoActor.AddUser(ctx, tableName, user)
    if err != nil {
        panic(err)
    }
}

// ListRecentLogEvents gets the most recent log stream and events for the
// specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
    functionName string) {
    log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
    helper.Pause(10)
    log.Println("Okay, let's check the logs to find what's happened recently with
    your Lambda function.")
    logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
    if err != nil {
        panic(err)
    }
    log.Printf("Getting some recent events from log stream %v\n",
        *logStream.LogStreamName)
```

```
events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
*logStream.LogStreamName, 10)
if err != nil {
    panic(err)
}
for _, event := range events {
    log.Printf("\t%v", *event.Message)
}
log.Println(strings.Repeat("-", 88))
}
```

Create a struct that wraps Amazon Cognito actions.

```
import (
    "context"
    "errors"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
    "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
    CognitoClient *cognitoidentityprovider.Client
}

// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
// trigger.
type Trigger int

const (
    PreSignUp Trigger = iota
    UserMigration
    PostAuthentication
)

type TriggerInfo struct {
```

```
Trigger      Trigger
HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
// is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
string, triggers ...TriggerInfo) error {
output, err := actor.CognitoClient.DescribeUserPool(ctx,
&cognitoidentityprovider.DescribeUserPoolInput{
    UserPoolId: aws.String(userPoolId),
})
if err != nil {
    log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
userPoolId, err)
    return err
}
lambdaConfig := output.UserPool.LambdaConfig
for _, trigger := range triggers {
    switch trigger.Trigger {
    case PreSignUp:
        lambdaConfig.PreSignUp = trigger.HandlerArn
    case UserMigration:
        lambdaConfig.UserMigration = trigger.HandlerArn
    case PostAuthentication:
        lambdaConfig.PostAuthentication = trigger.HandlerArn
    }
}
_, err = actor.CognitoClient.UpdateUserPool(ctx,
&cognitoidentityprovider.UpdateUserPoolInput{
    UserPoolId:    aws.String(userPoolId),
    LambdaConfig: lambdaConfig,
})
if err != nil {
    log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
}
return err
}

// SignUp signs up a user with Amazon Cognito.
```

```
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName string, password string, userEmail string) (bool, error) {
    confirmed := false
    output, err := actor.CognitoClient.SignUp(ctx,
        &cognitoIdentityProvider.SignUpInput{
            ClientId: aws.String(clientId),
            Password: aws.String(password),
            Username: aws.String(userName),
            UserAttributes: []types.AttributeType{
                {Name: aws.String("email"), Value: aws.String(userEmail)},
            },
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
        }
    } else {
        confirmed = output.UserConfirmed
    }
    return confirmed, err
}
```

```
// SignIn signs in a user to Amazon Cognito using a username and password authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName string, password string) (*types.AuthenticationResultType, error) {
    var authResult *types.AuthenticationResultType
    output, err := actor.CognitoClient.InitiateAuth(ctx,
        &cognitoIdentityProvider.InitiateAuthInput{
            AuthFlow:      "USER_PASSWORD_AUTH",
            ClientId:     aws.String(clientId),
            AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
        })
    if err != nil {
        var resetRequired *types.PasswordResetRequiredException
        if errors.As(err, &resetRequired) {
            log.Println(*resetRequired.Message)
        } else {
            log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
        }
    }
}
```

```
    }
} else {
    authResult = output.AuthenticationResult
}
return authResult, err
}

// ForgotPassword starts a password recovery flow for a user. This flow typically
// sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
    userName string) (*types.CodeDeliveryDetailsType, error) {
    output, err := actor.CognitoClient.ForgotPassword(ctx,
        &cognitoIdentityProvider.ForgotPasswordInput{
            ClientId: aws.String(clientId),
            Username: aws.String(userName),
        })
    if err != nil {
        log.Printf("Couldn't start password reset for user '%v'. Here's why: %v\n",
            userName, err)
    }
    return output.CodeDeliveryDetails, err
}

// ConfirmForgotPassword confirms a user with a confirmation code and a new
// password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
    string, code string, userName string, password string) error {
    _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
        &cognitoIdentityProvider.ConfirmForgotPasswordInput{
            ClientId:      aws.String(clientId),
            ConfirmationCode: aws.String(code),
            Password:      aws.String(password),
            Username:      aws.String(userName),
        })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
    
```

```
    log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
}
return err
}

// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
  string) error {
_, err := actor.CognitoClient.DeleteUser(ctx,
&cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
})
if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
}
return err
}

// AdminCreateUser uses administrator credentials to add a user to a user pool.
  This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
  string, userName string, userEmail string) error {
_, err := actor.CognitoClient.AdminCreateUser(ctx,
&cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:      aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
aws.String(userEmail)}},
})
if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
    log.Printf("User %v already exists in the user pool.", userName)
    err = nil
  } else {
    log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
}
```

```
}

return err
}

// AdminSetUserPassword uses administrator credentials to set a password for a
// user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
    string, userName string, password string) error {
    _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
    &cognitoidentityprovider.AdminSetUserPasswordInput{
        Password: aws.String(password),
        UserPoolId: aws.String(userPoolId),
        Username: aws.String(userName),
        Permanent: true,
    })
    if err != nil {
        var invalidPassword *types.InvalidPasswordException
        if errors.As(err, &invalidPassword) {
            log.Println(*invalidPassword.Message)
        } else {
            log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
err)
        }
    }
    return err
}
```

Create a struct that wraps DynamoDB actions.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodbattribute"
```

```
"github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type DynamoActions struct {
    DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
    UserName  string
    UserEmail string
    LastLogin *LoginInfo `dynamodbav:"",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
    UserPoolId string
    ClientId   string
    Time       string
}

// UserList defines a list of users.
type UserList struct {
    Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
// strings.
func (users *UserList) UserNameList() []string {
    names := make([]string, len(users.Users))
    for i := 0; i < len(users.Users); i++ {
        names[i] = users.Users[i].UserName
    }
    return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
    error {
    var err error
    var item map[string]types.AttributeValue
```

```
var writeReqs []types.WriteRequest
for i := 1; i < 4; i++ {
    item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
    if err != nil {
        log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
        return err
    }
    writeReqs = append(writeReqs, types.WriteRequest{PutRequest: &types.PutRequest{Item: item}})
}
_, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
    RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
})
if err != nil {
    log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName, err)
}
return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList, error) {
    var userList UserList
    output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
        TableName: aws.String(tableName),
    })
    if err != nil {
        log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
    } else {
        err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
        if err != nil {
            log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
        }
    }
    return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User) error {
```

```
userItem, err := attributevalue.MarshalMap(user)
if err != nil {
    log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
}
_, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
    Item:      userItem,
    TableName: aws.String(tableName),
})
if err != nil {
    log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
}
return err
}
```

Create a struct that wraps CloudWatch Logs actions.

```
import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
    "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
    CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
    functionName string) (types.LogStream, error) {
    var logStream types.LogStream
    logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
    output, err := actor.CwlClient.DescribeLogStreams(ctx,
        &cloudwatchlogs.DescribeLogStreamsInput{
            Descending: aws.Bool(true),
            Limit:      aws.Int32(1),
            LogGroupName: aws.String(logGroupName),
    })
    if err != nil {
        return logStream, err
    }
    logStream = output.Streams[0]
    return logStream, nil
}
```

```
    OrderBy:      types.OrderByLastEventTime,
})
if err != nil {
    log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
    logGroupName, err)
} else {
    logStream = output.LogStreams[0]
}
return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
// stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
string, logStreamName string, eventCount int32) (
[]types.OutputLogEvent, error) {
var events []types.OutputLogEvent
logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
output, err := actor.CwlClient.GetLogEvents(ctx,
&cloudwatchlogs.GetLogEventsInput{
    LogStreamName: aws.String(logStreamName),
    Limit:         aws.Int32(eventCount),
    LogGroupName:  aws.String(logGroupName),
})
if err != nil {
    log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
    logStreamName, err)
} else {
    events = output.Events
}
return events, err
}
```

Create a struct that wraps AWS CloudFormation actions.

```
import (
    "context"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
```

```
"github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
    CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
// structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
    string) StackOutputs {
    output, err := actor.CfnClient.DescribeStacks(ctx,
        &cloudformation.DescribeStacksInput{
            StackName: aws.String(stackName),
        })
    if err != nil || len(output.Stacks) == 0 {
        log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
            stackName, err)
    }
    stackOutputs := StackOutputs{}
    for _, out := range output.Stacks[0].Outputs {
        stackOutputs[*out.OutputKey] = *out.OutputValue
    }
    return stackOutputs
}
```

Clean up resources.

```
import (
    "context"
    "log"
    "user_pools_and_lambda_triggers/actions"

    "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
```

```
// cleanup when the example finishes.

type Resources struct {
    userPoolId      string
    userAccessTokens []string
    triggers        []actions.Trigger

    cognitoActor *actions.CognitoActions
    questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
demotools.IQuestioner) {
    resources.userAccessTokens = []string{}
    resources.triggers = []actions.Trigger{}
    resources.cognitoActor = cognitoActor
    resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            log.Printf("Something went wrong during cleanup.\n%v\n", r)
            log.Println("Use the AWS Management Console to remove any remaining resources
\n" +
                "that were created for this scenario.")
        }
    }()
}

wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
resources that were created "+
    "during this demo (y/n)?", "y")
if wantDelete {
    for _, accessToken := range resources.userAccessTokens {
        err := resources.cognitoActor.DeleteUser(ctx, accessToken)
        if err != nil {
            log.Println("Couldn't delete user during cleanup.")
            panic(err)
        }
        log.Println("Deleted user.")
    }
    triggerList := make([]actions.TriggerInfo, len(resources.triggers))
    for i := 0; i < len(resources.triggers); i++ {
```

```
    triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],  
    HandlerArn: nil}  
  }  
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,  
  triggerList...)  
  if err != nil {  
    log.Println("Couldn't update Cognito triggers during cleanup.")  
    panic(err)  
  }  
  log.Println("Removed Cognito triggers from user pool.")  
} else {  
  log.Println("Be sure to remove resources when you're done with them to avoid  
unexpected charges!")  
}  
}  
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
 - [AdminCreateUser](#)
 - [AdminSetUserPassword](#)
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [UpdateUserPool](#)

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples for Amazon Cognito Sync using AWS SDKs

The following code examples show how to use Amazon Cognito Sync with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Basic examples for Amazon Cognito Sync using AWS SDKs](#)
 - [Actions for Amazon Cognito Sync using AWS SDKs](#)
 - [Use ListIdentityPoolUsage with an AWS SDK](#)

Basic examples for Amazon Cognito Sync using AWS SDKs

The following code examples show how to use the basics of Amazon Cognito Sync with AWS SDKs.

Examples

- [Actions for Amazon Cognito Sync using AWS SDKs](#)
 - [Use ListIdentityPoolUsage with an AWS SDK](#)

Actions for Amazon Cognito Sync using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Cognito Sync actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the [Amazon Cognito Sync API Reference](#).

Examples

- [Use ListIdentityPoolUsage with an AWS SDK](#)

Use ListIdentityPoolUsage with an AWS SDK

The following code example shows how to use ListIdentityPoolUsage.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn show_pools(client: &Client) -> Result<(), Error> {
    let response = client
        .list_identity_pool_usage()
        .max_results(10)
        .send()
        .await?;

    let pools = response.identity_pool_usages();
    println!("Identity pools:");

    for pool in pools {
        println!(
            "  Identity pool ID:      {}",
            pool.identity_pool_id().unwrap_or_default()
        );
        println!(
            "  Data storage:          {}",
            pool.data_storage().unwrap_or_default()
        );
        println!(
            "  Sync sessions count:  {}",
            pool.sync_sessions_count().unwrap_or_default()
        );
        println!(
            "  Last modified:         {}",
            pool.last_modified_date().unwrap().to_chrono_utc()?
        );
        println!();
    }

    println!("Next token: {}", response.next_token().unwrap_or_default());
}
```

```
    Ok(()  
}
```

- For API details, see [ListIdentityPoolUsage](#) in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Multi-tenant application best practices

Amazon Cognito user pools operate with multi-tenant applications that generate a volume of requests that must remain within Amazon Cognito quotas. To scale up this capacity when your customer base grows, you can [purchase additional quota capacity](#).

Note

Amazon Cognito [quotas](#) are applied per AWS account and AWS 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.

This section describes methods that you can implement to separate tenants between Amazon Cognito resources within the same Region and AWS account. You can also split your tenants across more than one AWS account or Region, and give each of them their own quota. Other advantages of multi-Region multi-tenancy include the highest possible level of isolation, shortest network-transit time for globally distributed users, and adherence to existing distribution models in your organization.

Single-Region multi-tenancy can also have advantages for your customers and administrators.

The following list covers some of the advantages of multi-tenancy with shared resources.

Advantages of multi-tenancy

Common user directory

Multi-tenancy supports models where customers have accounts in more than one application. You can [link identities from third-party providers](#) into a single consistent user pool profile. In cases where user profiles are unique to their tenant, any multi-tenancy strategy with a single user pool has one point of entry to user administration.

Common security

In a shared user pool, you can create a single standard for security and apply the same [threat protection](#), [multi-factor authentication](#) (MFA), and [AWS WAF](#) standards to all tenants. Because an AWS WAF web ACL must be in the same AWS Region as the resource that you associate it with, multi-tenancy offers shared access to a complex resource. When you want to maintain

consistent security configuration in multi-Region Amazon Cognito applications, you must apply operational standards that replicate your configuration between resources.

Common customization

You can customize user pools and identity pools with AWS Lambda. Configuration of [Lambda triggers](#) in user pools and [Amazon Cognito events](#) in identity pools can become complex.

Lambda functions must be in the same AWS Region as your user pool or identity pool. Shared Lambda functions can enforce standards for custom authentication flows, user migration, token generation, and other functions within a Region.

Common messaging

Amazon Simple Notification Service (Amazon SNS) requires additional configuration in a Region before you can send [SMS messages](#) to your users. You can send [email messages](#) with Amazon Simple Email Service (Amazon SES) verified identities and domains that are contained within a Region.

With multi-tenancy, you can share this configuration and maintenance overhead between all of your tenants. Because Amazon SNS and Amazon SES aren't available in all AWS Regions, splitting your resources between Regions requires additional consideration.

When you use [custom messaging providers](#), you gain the *common customization* of a single Lambda function to manage your message delivery.

[Managed login](#) sets a session cookie in the browser so that it recognizes a user who has already authenticated. When you authenticate *local users* in a user pool, their session cookie authenticates them for all app clients in the same user pool. A local user exists exclusively in your user pool directory without federation through an external IdP. The session cookie is valid for one hour. You can't change the session cookie duration.

There are two ways to prevent sign-in across app clients with a hosted UI session cookie.

- Separate your users into per-tenant user pools.
- Replace hosted UI sign-in with Amazon Cognito user pools API sign-in.

Topics

- [User-pool multi-tenancy best practices](#)
- [App-client multi-tenancy best practices](#)

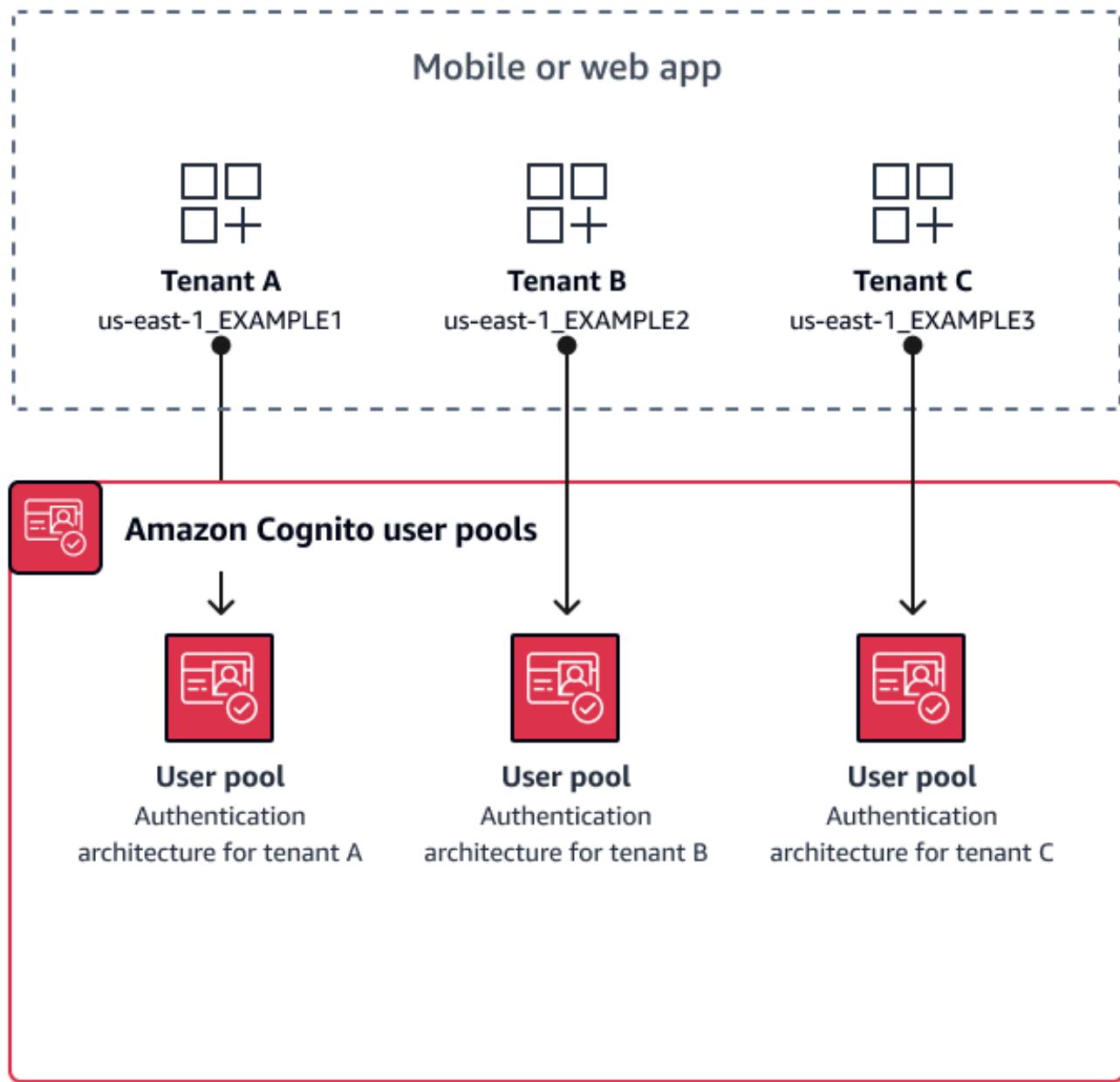
- [User group multi-tenancy best practices](#)
- [Custom-attribute multi-tenancy best practices](#)
- [Custom scope multi-tenancy best practices](#)
- [Multi-tenancy security recommendations](#)

User-pool multi-tenancy best practices

Create a user pool for each tenant in your app. This approach provides maximum isolation for each tenant. You can implement different configurations for each tenant. Tenant isolation by user pool gives you flexibility in user-to-tenant mapping. You can create multiple profiles for the same user. However, each user must sign up individually for each tenant they can access.

Using this approach, you can set up a hosted UI for each tenant independently and redirect users to their tenant-specific instance of your application. You can also use this approach to integrate with backend services like [Amazon API Gateway](#).

The following diagram shows each tenant with a dedicated user pool.



When to implement user-pool multi-tenancy

When isolation and customization are your primary concerns. The relationship between users and tenants might be complex in an architecture with multiple user pools. Consider an example where you have two educational tenants. The same user might be a limited-access student in one app, and a teacher with a high level of permissions in another. You might require MFA in one app but not another, or have a different password policy. Because local users can sign in to multiple app

clients in user pools with managed login, user-pool multi-tenancy is also ideal when you want more than one of your tenants to sign in with managed login.

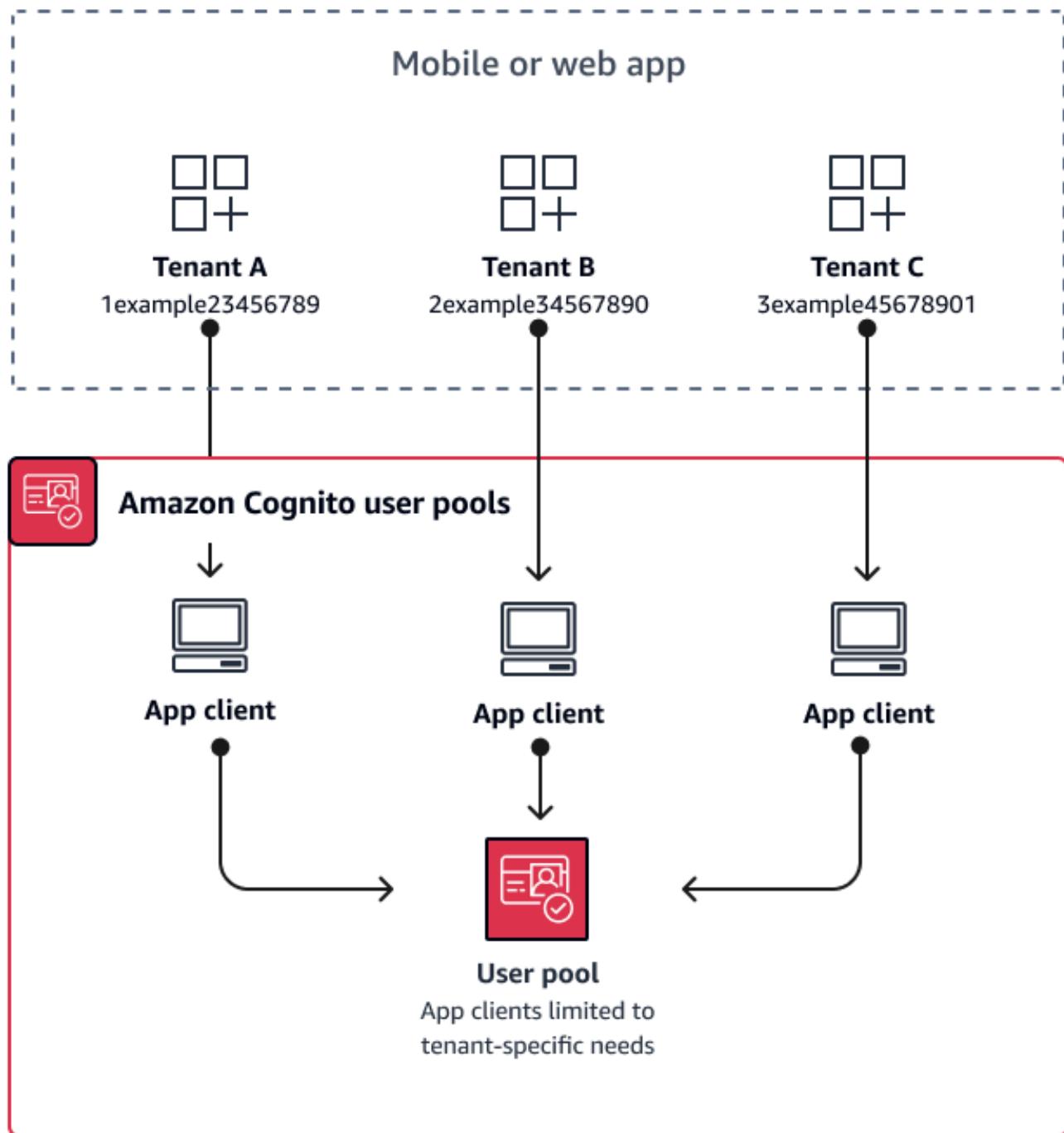
Level of effort

The development and operation effort to use this approach is high. To ensure consistent and predictable outcomes for your family of apps, you must integrate Amazon Cognito resources with your automation tools and maintain your baselines as your authentication architecture grows more complex. When you want to create a single starting place for your apps, you have to build the user-interface (UI) elements to capture the initial decision that routes users to the correct resource.

App-client multi-tenancy best practices

Create an [app client](#) for each tenant in your app. With app-client multi-tenancy, you can assign any user to tenant-linked app clients and retain a single user profile. Because you can assign any or all of the [identity providers \(IdPs\)](#) in your user pool to an app client, a tenant app client can permit sign-in with a tenant-specific IdP. When users exist in multiple tenants, you can link their profiles with multiple IdPs for a consistent user experience.

The following diagram shows each tenant with a dedicated app client in a shared user pool.



When to implement app-client multi-tenancy

When you can choose a universal configuration for settings at the user-pool level, like Lambda triggers, password policy, and the content and delivery methods of email and SMS messages.

Because users in a shared user pool can sign in to any app client, app-client multi-tenancy is ideal for sign-in with app-client-specific IdPs or the Amazon Cognito user pools API. App-client multi-tenancy is also well-suited for one-to-many environments where you want to permit users to transition between multiple applications.

Level of effort

App-client multi-tenancy requires moderate effort. A major challenge of app-client multi-tenancy is the ability for tenants to present a hosted UI cookie and switch between apps. In an app-client multi-tenancy architecture, avoid hosted UI sign-in where isolation is necessary. You can distribute your mobile app or links to your web app with app client logic built in, or you can build initial UI elements that determine users' tenancy. The level of effort is lower because you don't need to standardize and maintain configuration across multiple user pools and identity pools.

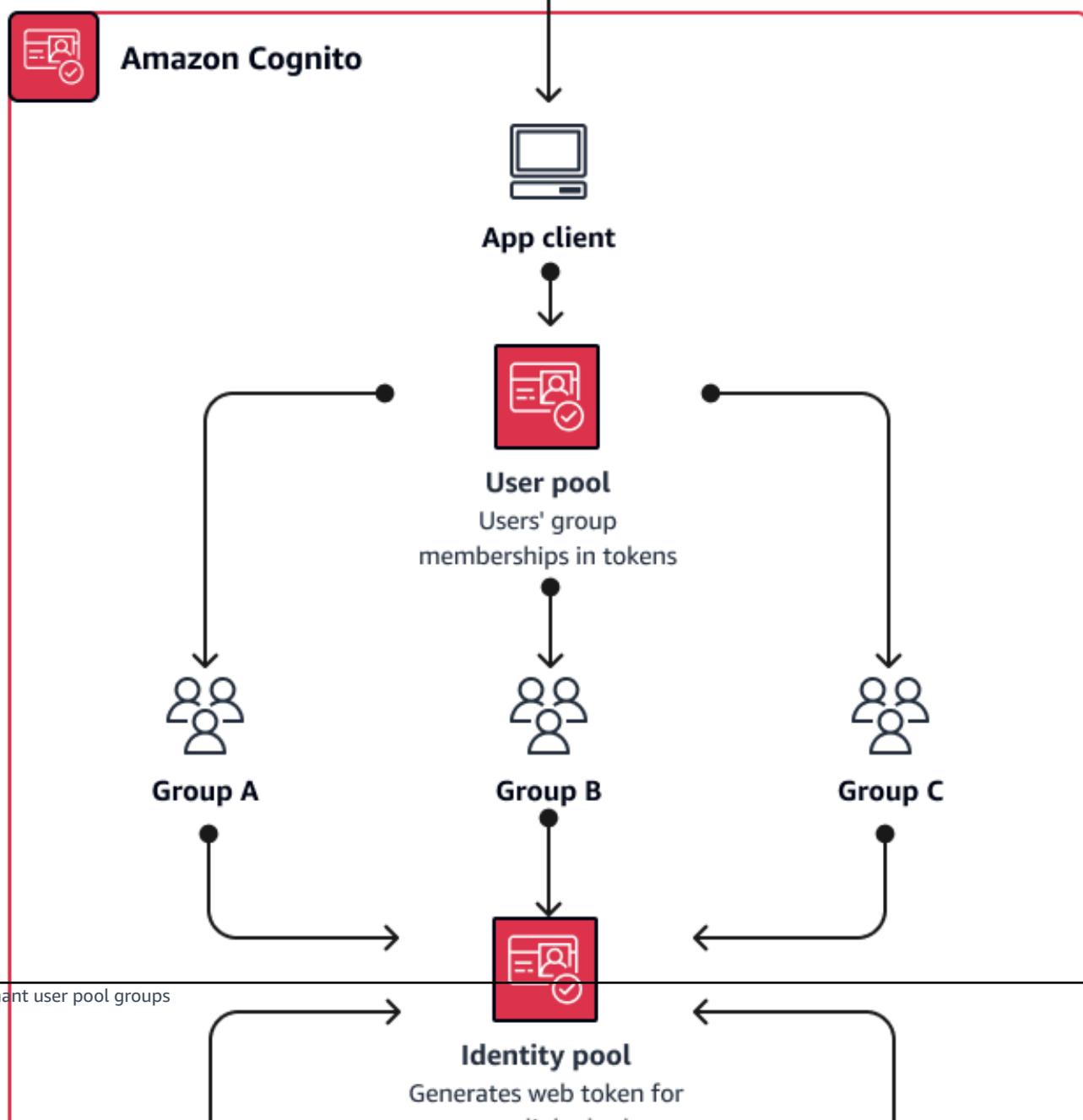
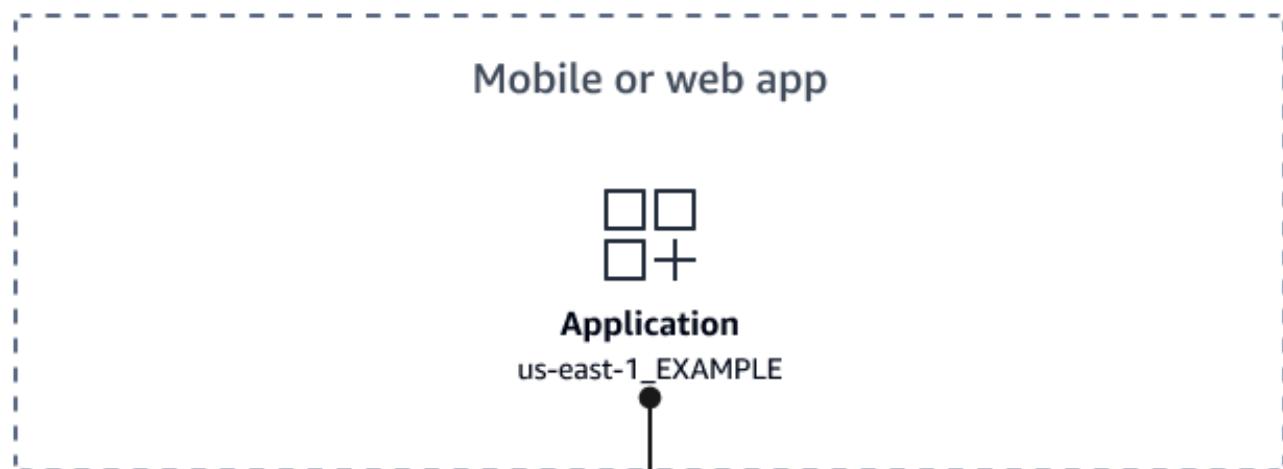
User group multi-tenancy best practices

Group-based multi-tenancy works best when your architecture requires Amazon Cognito user pools with identity pools.

User pool [ID and access tokens](#) contain a `cognito:groups` claim. Additionally, ID tokens contain `cognito:roles` and `cognito:preferred_role` claims. When the primary outcome of authentication in your app is temporary AWS credentials from an identity pool, your users' group memberships can determine the [IAM role](#) and permissions that they receive.

As an example, consider three tenants that each store application assets in their own Amazon S3 bucket. Assign the users of each tenant to an associated group, configure a preferred role for the group, and grant that role read access to their bucket.

The following diagram shows tenants sharing an app client and a user pool, with dedicated groups in the user pool that determine their eligibility for an IAM role.



When to implement group multi-tenancy

When access to AWS resources is your primary concern. Groups in Amazon Cognito user pools are a mechanism for role-based access control (RBAC). You can configure many groups in a user pool and make complex RBAC decisions with group priority. Identity pools can assign credentials for the role with the highest priority, any role in the groups claim, or from other claims in a user's tokens.

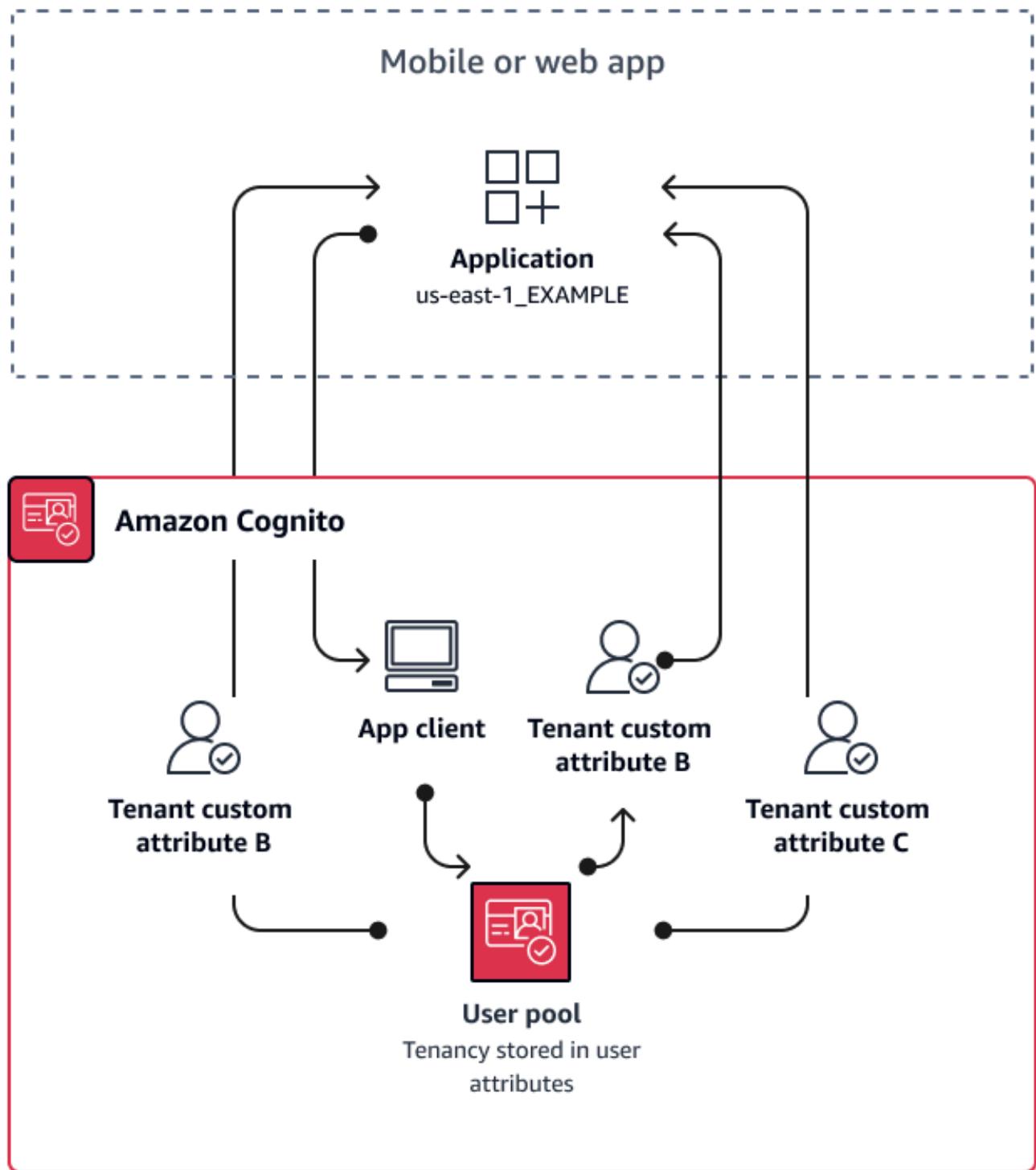
Level of effort

The level of effort to maintain multi-tenancy with group membership alone is low. However, to expand the role of user pool groups beyond the built-in capacity for IAM role selection, you must build application logic that processes group membership in users' tokens, and determine what to do in the client. You can integrate Amazon Verified Permissions with your apps to make client-side authorization decisions. Group identifiers aren't currently processed in Verified Permissions [IsAuthorizedWithToken](#) API operations, but you can [develop custom code](#) that parses the contents of tokens, including group-membership claims.

Custom-attribute multi-tenancy best practices

Amazon Cognito supports [custom attributes](#) with names that you choose. One scenario where custom attributes are useful is when they distinguish the tenancy of users in a shared user pool. When you assign users a value for an attribute like `custom:tenantID`, your app can assign access to tenant-specific resources accordingly. A custom attribute that defines a tenant ID should be immutable or read-only to the app client.

The following diagram shows tenants sharing an app client and a user pool, with a custom attributes in the user pool that indicates the tenant that they belong to.



When custom attributes determine tenancy, you can distribute a single application or sign-in URL. After your user signs in, your app can process the `custom:tenantID` claim to determine which assets

to load, the branding to apply, and features to display. For advanced access-control decisions from user attributes, set up your user pool as an identity provider in Amazon Verified Permissions, and generate access decisions from the contents of ID or access tokens.

When to implement custom-attribute multi-tenancy

When tenancy is surface-level. A tenant attribute can contribute to branding and layout outcomes. When you want to achieve significant isolation between tenants, custom attributes aren't the best choice. Any difference between tenants that must be configured at the user-pool or app-client level, like MFA or hosted UI branding, requires that you create distinctions between tenants in a way that custom attributes don't offer. With identity pools, you can even choose the IAM role from your users from the custom-attribute claim in their ID token.

Level of effort

Because custom-attribute multi-tenancy transfers the duty of tenant-based authorization decisions on your app, the level of effort tends to be high. If you're already well-versed in a client configuration that parses OIDC claims, or in Amazon Verified Permissions, this approach might require the lowest level of effort.

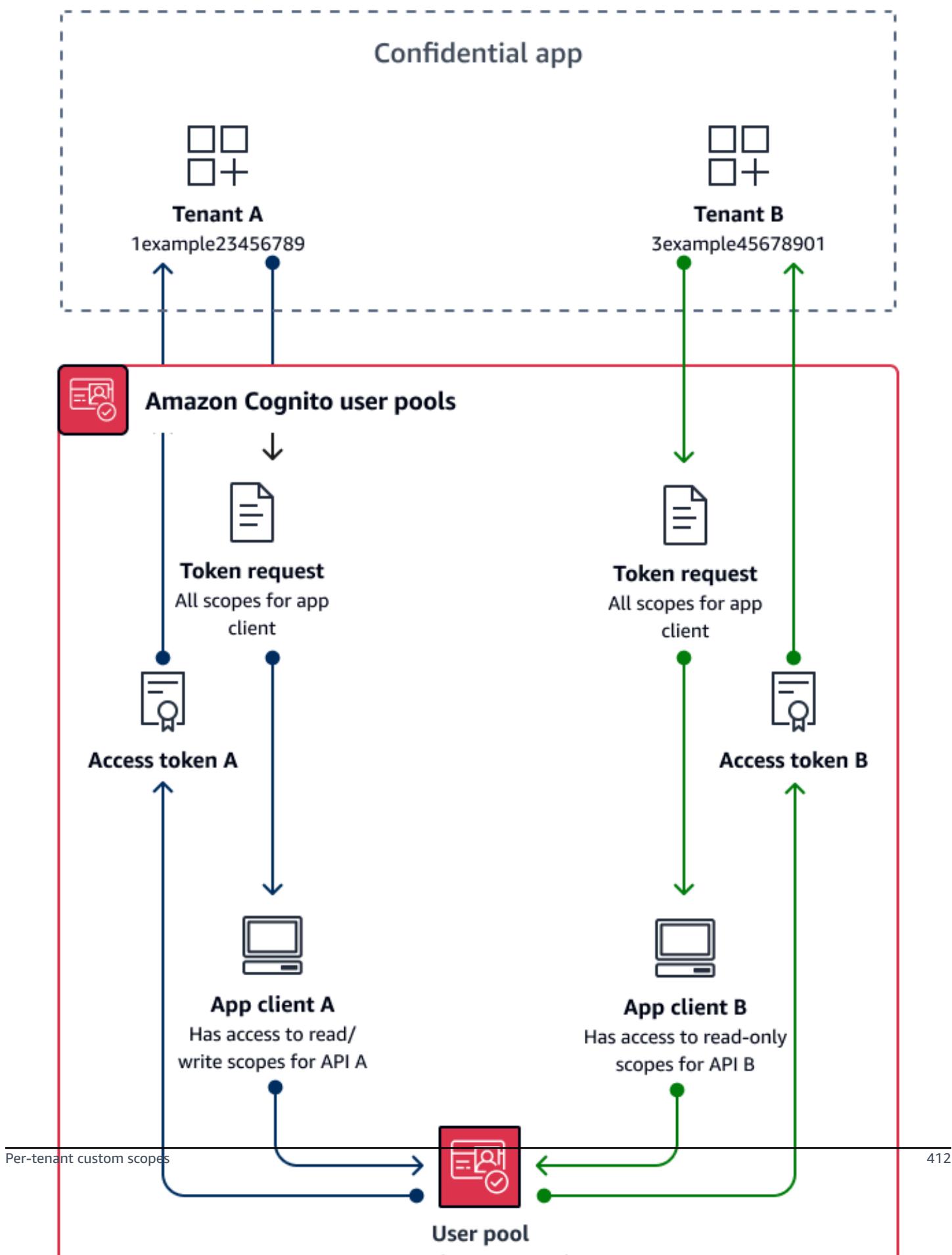
Custom scope multi-tenancy best practices

Amazon Cognito supports custom OAuth 2.0 scopes for [resource servers](#). You can implement app client multi-tenancy in users pools for machine-to-machine (M2M) authorization models with custom scopes. Scope-based multi-tenancy reduces the effort required to implement M2M multi-tenancy by defining access in your app client or application configuration.

Note

Currently, you can't [customize access tokens](#) to add custom claims or scopes in client credentials (M2M) authorization flows.

The following diagram illustrates one option for custom scope multi-tenancy. It shows each tenant with a dedicated app client that has access to relevant scopes in a user pool.



When to implement custom-scope multi-tenancy

When your usage is M2M authorization with client credentials in a confidential client. As a best practice, create resource servers that are exclusive to an app client. Custom scope multi-tenancy can be *request-dependent* or *client-dependent*.

Request-dependent

Implement application logic to request only the scopes that match the requirements of your tenant. For example, an app client might be able to issue read and write access to API A and API B, but tenant application A requests only the read scope for API A and the scope that indicates tenancy. This model allows for more complex combinations of shared scopes between tenants.

Client-dependent

Request all scopes assigned to an app client in your authorization requests. To do this, omit the scope request parameter in your request to the [Token endpoint](#). This model allows for app clients to store the access indicators that you want to add to your custom scopes.

In either case, your applications receive access tokens with scopes that indicate their privileges for data sources that they depend on. Scopes can also present other information to your application:

- Designate tenancy
- Contribute to request logging
- Indicate the APIs that the application is authorized to query
- Inform initial checks for active customers.

Level of effort

Custom-scope multi-tenancy requires a varying level of effort relative to the scale of your application. You must devise application logic that allows your applications to parse access tokens and make the appropriate API requests.

For example, a resource server scope comes in the format [resource server identifier]/[name]. The resource server identifier is unlikely to be relevant to the authorization decision from the tenant scope, requiring the scope name to be consistently parsed.

Example resource

The following AWS CloudFormation template creates a user pool for custom-scope multi-tenancy with one resource server and app client.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: A sample template illustrating scope-based multi-tenancy
Resources:
  MyUserPool:
    Type: "AWS::Cognito::UserPool"
  MyUserPoolDomain:
    Type: AWS::Cognito::UserPoolDomain
    Properties:
      UserPoolId: !Ref MyUserPool
      # Note that the value for "Domain" must be unique across all of AWS.
      # In production, you may want to consider using a custom domain.
      # See: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-add-custom-domain.html#cognito-user-pools-add-custom-domain-adding
      Domain: !Sub "example-userpool-domain-${AWS::AccountId}"
  MyUserPoolResourceServer:
    Type: "AWS::Cognito::UserPoolResourceServer"
    Properties:
      Identifier: resource1
      Name: resource1
      Scopes:
        - ScopeDescription: Read-only access
          ScopeName: readScope
      UserPoolId: !Ref MyUserPool
  MyUserPoolTenantBatch1ResourceServer:
    Type: "AWS::Cognito::UserPoolResourceServer"
    Properties:
      Identifier: TenantBatch1
      Name: TenantBatch1
      Scopes:
        - ScopeDescription: tenant1 identifier
          ScopeName: tenant1
        - ScopeDescription: tenant2 identifier
          ScopeName: tenant2
      UserPoolId: !Ref MyUserPool
  MyUserPoolClientTenant1:
    Type: "AWS::Cognito::UserPoolClient"
    Properties:
      AllowedOAuthFlows:
```

```
- client_credentials
AllowedOAuthFlowsUserPoolClient: true
AllowedOAuthScopes:
  - !Sub "${MyUserPoolTenantBatch1ResourceServer}/tenant1"
  - !Sub "${MyUserPoolResourceServer}/readScope"
GenerateSecret: true
UserPoolId: !Ref MyUserPool
Outputs:
  UserPoolClientId:
    Description: User pool client ID
    Value: !Ref MyUserPoolClientTenant1
  UserPoolDomain:
    Description: User pool domain
    Value: !Sub "https://${MyUserPoolDomain}.auth.${AWS::Region}.amazoncognito.com"
```

Multi-tenancy security recommendations

To help make your application more secure, we recommend the following:

- Validate tenancy in your app with Amazon Verified Permissions. Build policies that examine user pool, app client, group, or custom-attribute entitlement before you permit a user's request in your application. AWS created Verified Permissions [identity sources](#) with Amazon Cognito user pools in mind. Verified Permissions has [additional guidance](#) for multi-tenancy management.
- Use only a verified email address to authorize user access to a tenant based on domain match. Do not trust email addresses and phone numbers unless your app verifies them, or the external IdP gives a proof of verification. For more details on setting these permissions, see [Attribute Permissions and Scopes](#).
- Use *immutable*, or read-only, custom attributes for the user profile attributes that identify tenants. You can only set the value of immutable attributes when you create a user or a user signs up in your user pool. Also, give app clients read-only access to the attributes.
- Use 1:1 mapping between a tenant's external IdP and application client to prevent unauthorized cross-tenant access. A user who has been authenticated by an external IdP, and who has a valid Amazon Cognito session cookie, can access other tenant apps that trust the same IdP.
- When you implement tenant-matching and authorization logic in your application, restrict users so that they can't modify the criteria that authorize user access to the tenants. Also, if an external IdP is being used for federation, restrict tenant identity provider administrators so that they can't modify user access.

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 temporary 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 either sign in directly through a user pool, or they can 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](#)
- [Access back-end resources with user pool tokens](#)
- [Access resources with API Gateway and Lambda with a user pool](#)
- [Access AWS services with a user pool and an identity pool](#)
- [Authenticate with a third party and access AWS services with an identity pool](#)
- [Access AWS AppSync resources with Amazon Cognito](#)

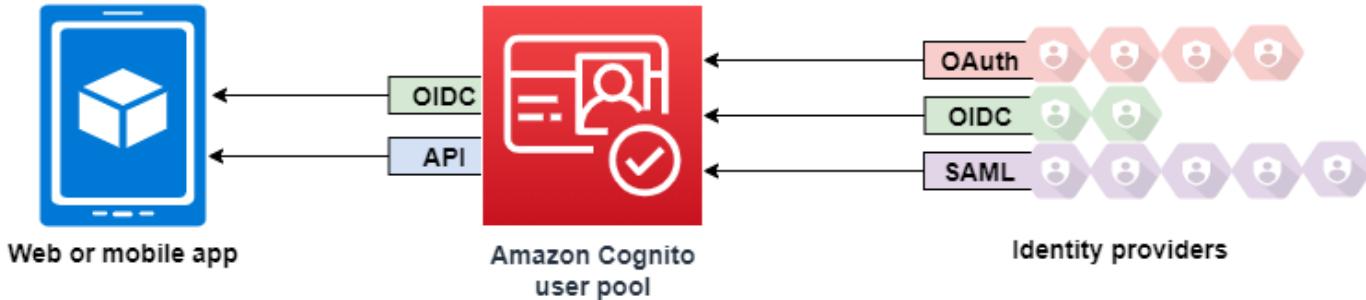
Authenticate with a user pool

You can enable your users to authenticate with a user pool. Your app users can either sign in directly through a user pool, or they can 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

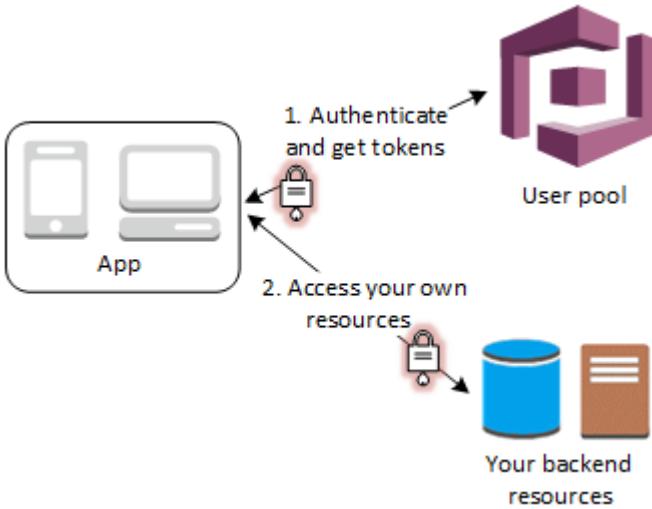
access other AWS services, or you might choose to use them to control access to your server-side resources, or to the Amazon API Gateway.

For more information, see [An example authentication session](#) and [Understanding user pool JSON web tokens \(JWTs\)](#).



Access back-end resources with user pool tokens

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 to your resources, see [Adding groups to a user pool](#).



After 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 [Scopes, M2M, and APIs with resource servers](#).

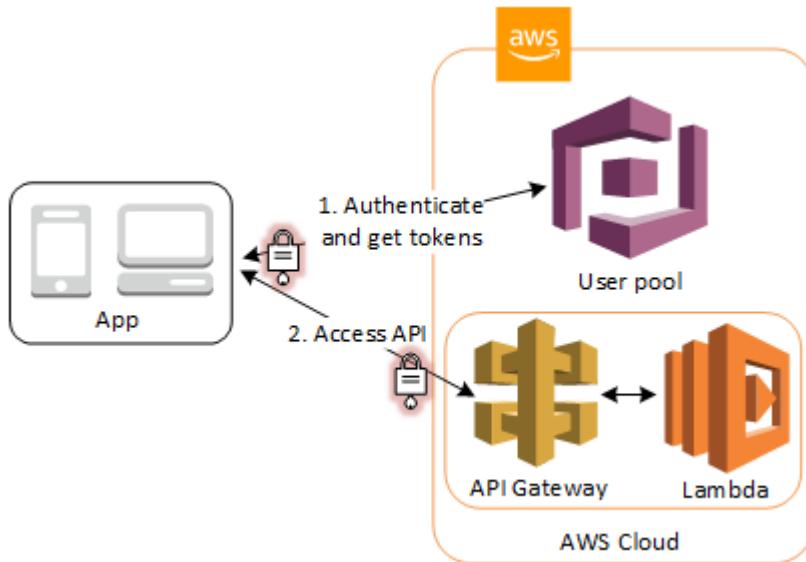
For more information about user pool authentication, see [An example authentication session](#) and [Understanding user pool JSON web tokens \(JWTs\)](#).

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.

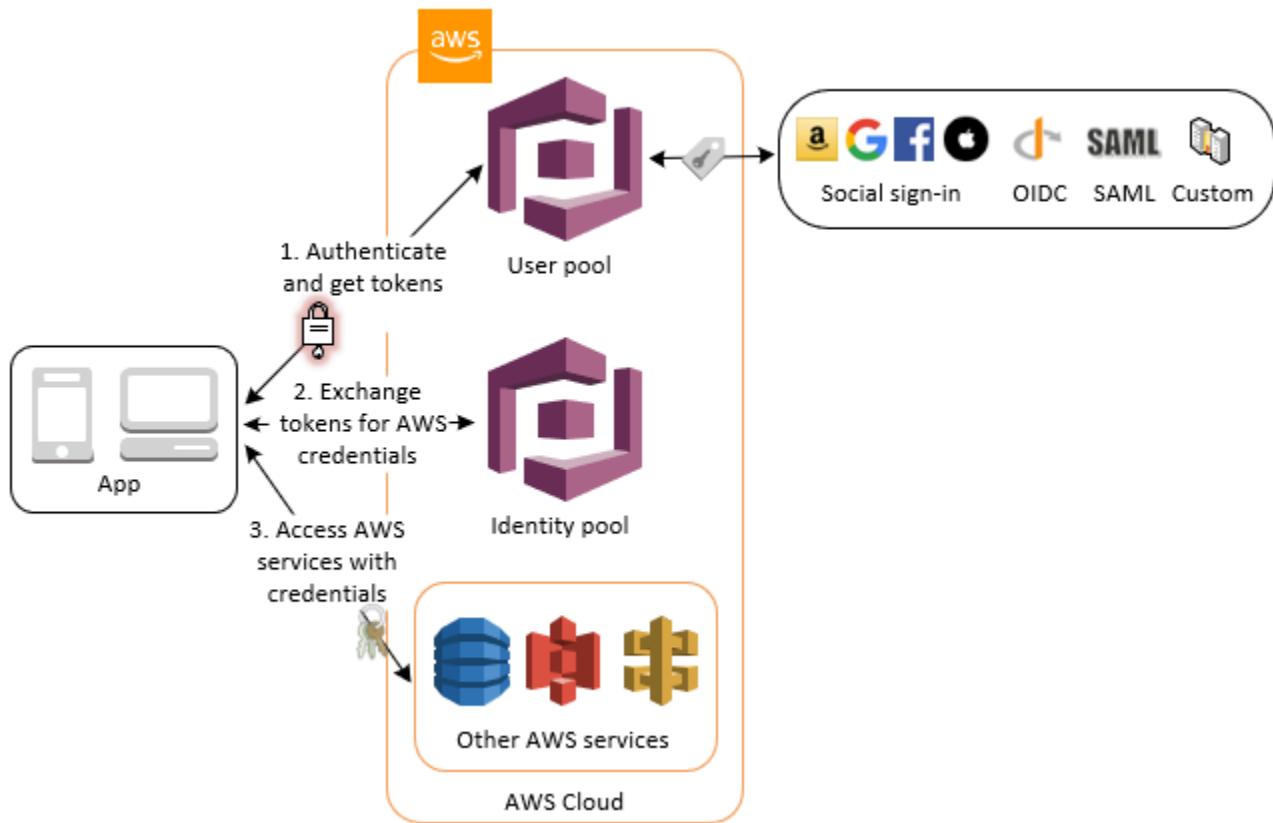
You can use groups in a user pool to control permissions with API Gateway by mapping group membership to IAM roles. The groups that a user is a member of are included in the ID token provided by a user pool when your app user signs in. For more information on user pool groups See [Adding groups to a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

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](#).



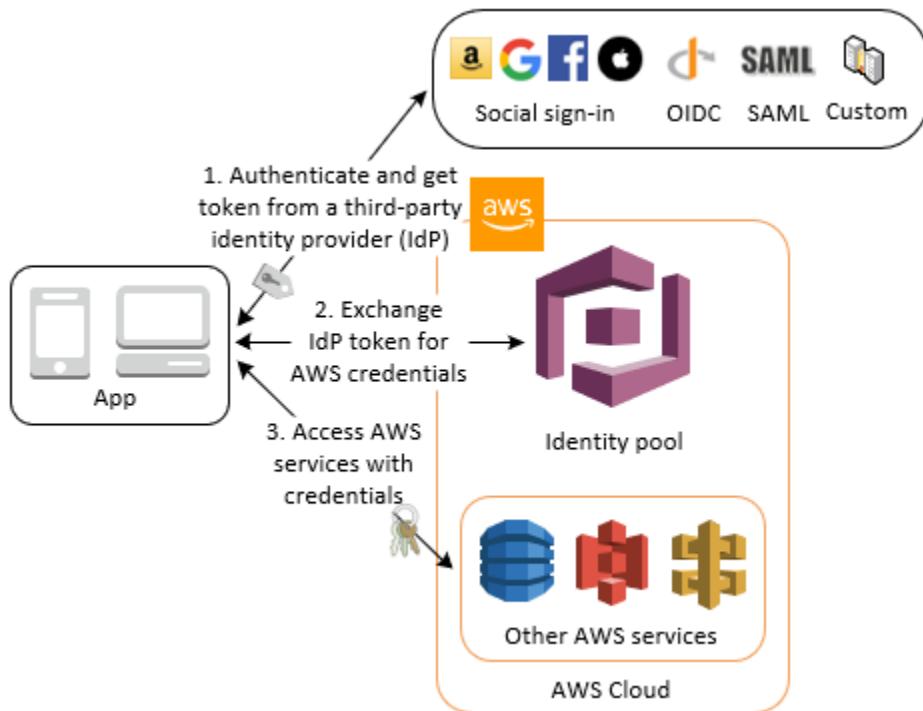
Access AWS services with a user pool and an identity pool

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](#) and [Getting started with Amazon Cognito identity pools](#).



Authenticate with a third party and access AWS services with an identity pool

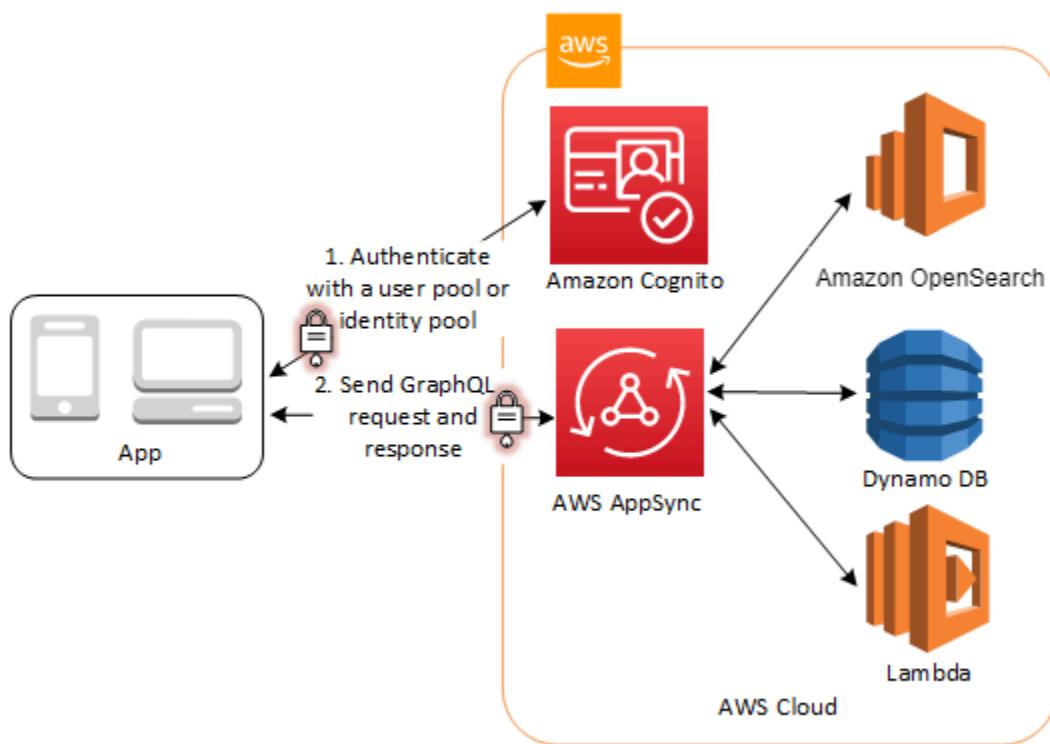
You can enable your users access to AWS services through an identity pool. An identity pool requires an IdP token from a user that's authenticated by a third-party identity provider (or nothing if it's an anonymous guest). In exchange, the identity pool grants temporary AWS credentials that you can use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



Access AWS AppSync resources with Amazon Cognito

You can grant your users access to AWS AppSync resources with tokens from a successful Amazon Cognito user pool authentication. For more information, see [AMAZON_COGNITO_USER_POOLS authorization](#) in the *AWS AppSync Developer Guide*.

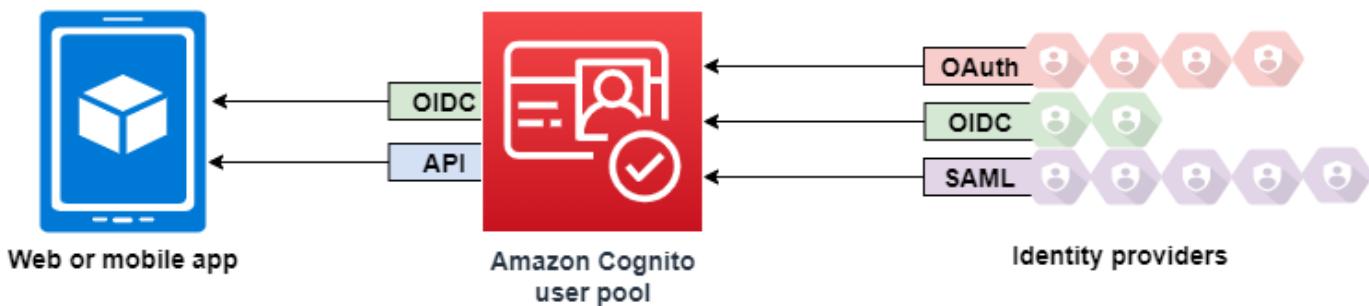
You can also sign requests to the AWS AppSync GraphQL API with the IAM credentials that you receive from an identity pool. See [AWS_IAM authorization](#).



Amazon Cognito user pools

An Amazon Cognito user pool is a user directory for web and mobile app authentication and authorization. From the perspective of your app, an Amazon Cognito user pool is an OpenID Connect (OIDC) identity provider (IdP). A user pool adds layers of additional features for security, identity federation, app integration, and customization of the user experience.

You can, for example, verify that your users' sessions are from trusted sources. You can combine the Amazon Cognito directory with an external identity provider. With your preferred AWS SDK, you can choose the API authorization model that works best for your app. And you can add AWS Lambda functions that modify or overhaul the default behavior of Amazon Cognito.



Topics

- [Features](#)
- [User pool feature plans](#)
- [Authentication with Amazon Cognito user pools](#)
- [User pool sign-in with a third party identity provider](#)
- [User pool managed login](#)
- [Customizing user pool workflows with Lambda triggers](#)
- [Managing users in your user pool](#)
- [Understanding user pool JSON web tokens \(JWTs\)](#)
- [Accessing resources after successful sign-in](#)
- [Configure user pool features](#)
- [Using Amazon Cognito user pools security features](#)
- [User pool endpoints and managed login reference](#)

Features

Amazon Cognito user pools have the following features.

Sign-up

Amazon Cognito user pools have user-driven, administrator-driven, and programmatic methods to add user profiles to your user pool. Amazon Cognito user pools supports the following sign-up models. You can use any combination of these models in your app.

Important

If you activate user sign-up in your user pool, anyone on the internet can sign up for an account and sign into your apps. Don't enable self-registration in your user pool unless you want to open your app to public sign-up. To change this setting, update **Self-service sign-up** in the **Sign-up** menu under **Authentication** in the user pool console, or update the value of [AllowAdminCreateUserOnly](#) in a [CreateUserPool](#) or [UpdateUserPool](#) API request. For information about security features that you can set up in your user pools, see [Using Amazon Cognito user pools security features](#).

1. Your users can enter their information in your app and create a user profile that's native to your user pool. You can call API sign-up operations to register users in your user pool. You can open these sign-up operations to anyone, or you can authorize them with a client secret or AWS credentials.
2. You can redirect users to a third-party IdP that they can authorize to pass their information to Amazon Cognito. Amazon Cognito processes OIDC id tokens, OAuth 2.0 userInfo data, and SAML 2.0 assertions into user profiles in your user pool. You control the attributes that you want Amazon Cognito to receive based on attribute-mapping rules.
3. You can skip public or federated sign-up, and create users based on your own data source and schema. Add users directly in the Amazon Cognito console or API. Import users from a CSV file. Run a just-in-time AWS Lambda function that looks up your new user in an existing directory, and populates their user profile from existing data.

After your users sign up, you can add them to groups that Amazon Cognito lists in the access and ID tokens. You can also link user pool groups to IAM roles when you pass the ID token to an identity pool.

Related topics

- [Managing users in your user pool](#)
- [Understanding API, OIDC, and managed login pages authentication](#)
- [Code examples for Amazon Cognito Identity Provider using AWS SDKs](#)

Sign-in

Amazon Cognito can be a standalone user directory and identity provider (IdP) to your app. Your users can sign in with managed login pages that are hosted by Amazon Cognito, or with a custom-built user authentication service through the Amazon Cognito user pools API. The application tier behind your custom-built front end can authorize requests on the back end with any of several methods to confirm legitimate requests.

Users can set up and sign in with usernames and passwords, passkeys, and email and SMS message one-time passwords. You can offer consolidate sign in with external user directories, multi-factor authentication (MFA) after sign-in, trust remembered devices, and custom authentication flows that you design.

To sign in users with an external directory, optionally combined with the user directory built in to Amazon Cognito, you can add the following integrations.

1. Sign in and import customer user data with OAuth 2.0 social sign-in. Amazon Cognito supports sign-in with Google, Facebook, Amazon, and Apple through OAuth 2.0.
2. Sign in and import work and school user data with SAML and OIDC sign-in. You can also configure Amazon Cognito to accept claims from any SAML or OpenID Connect (OIDC) identity provider (IdP).
3. Link external user profiles to native user profiles. A linked user can sign in with a third-party user identity and receive access that you assign to a user in the built-in directory.

Related topics

- [User pool sign-in with a third party identity provider](#)
- [Linking federated users to an existing user profile](#)

Machine-to-machine authorization

Some sessions aren't a human-to-machine interaction. You might need a service account that can authorize a request to an API by an automated process. To generate access tokens for machine-to-machine authorization with OAuth 2.0 scopes, you can add an app client that generates [client-credentials grants](#).

Related topics

- [Scopes, M2M, and APIs with resource servers](#)

Managed login

When you don't want to build a user interface, you can present your users with a customized managed login pages. Managed login is a set of web pages for sign-up, sign-in, multi-factor authentication (MFA), and password reset. You can add managed login to your existing domain, or use a prefix identifier in an AWS subdomain.

Related topics

- [User pool managed login](#)
- [Configuring a user pool domain](#)

Security

Your local users can provide an additional authentication factor with a code from an SMS or email message, or an app that generates multi-factor authentication (MFA) codes. You can build mechanisms to set up and process MFA in your application, or you can let managed login manage it. Amazon Cognito user pools can bypass MFA when your users sign in from trusted devices.

If you don't want to initially require MFA from your users, you can require it conditionally. With advanced security features, Amazon Cognito can detect potential malicious activity and require your user to set up MFA, or block sign-in.

If network traffic to your user pool might be malicious, you can monitor it and take action with AWS WAF web ACLs.

Related topics

- [Adding MFA to a user pool](#)
- [Advanced security with threat protection](#)

- [Associating an AWS WAF web ACL with a user pool](#)

Custom user experience

At most stages of a user's sign-up, sign-in, or profile update, you can customize how Amazon Cognito handles the request. With Lambda triggers, you can modify an ID token or reject a sign-up request based on custom conditions. You can create your own custom authentication flow.

You can upload custom CSS and logos to give managed login a familiar look and feel to your users.

Related topics

- [Customizing user pool workflows with Lambda triggers](#)
- [Custom authentication challenge Lambda triggers](#)
- [Apply branding to managed login pages](#)

Monitoring and analytics

Amazon Cognito user pools log API requests, including requests to managed login, to AWS CloudTrail. You can review performance metrics in Amazon CloudWatch Logs, push custom logs to CloudWatch with Lambda triggers, monitor email and SMS message delivery, and monitor API request volume in the Service Quotas console.

With the Plus [feature plan](#), you can monitor user authentication attempts for indicators of compromise with automated-learning technology and immediately remediate risks. These advanced security features also log user activity to your user pool and optionally, to Amazon S3, CloudWatch Logs, or Amazon Data Firehose.

You can also log device and session data from your API requests to an Amazon Pinpoint campaign. With Amazon Pinpoint, you can send push notifications from your app based on your analysis of user activity.

Related topics

- [Amazon Cognito logging in AWS CloudTrail](#)
- [Tracking quotas and usage in CloudWatch and Service Quotas](#)
- [Exporting logs from Amazon Cognito user pools](#)

- [Using Amazon Pinpoint for user pool analytics](#)

Amazon Cognito identity pools integration

The other half of Amazon Cognito is identity pools. Identity pools provide credentials that authorize and monitor API requests to AWS services, for example Amazon DynamoDB or Amazon S3, from your users. You can build identity-based access policies that protect your data based on how you classify the users in your user pool. Identity pools can also accept tokens and SAML 2.0 assertions from a variety of identity providers, independently of user pool authentication.

Related topics

- [Accessing AWS services using an identity pool after sign-in](#)
- [Amazon Cognito identity pools](#)

User pool feature plans

Understanding the cost is a crucial step in preparing to implement Amazon Cognito user pools authentication. Amazon Cognito has feature plans for user pools. Each plan has a set of features and a monthly cost per active user. Each feature plan unlocks access to more features than the one before it.

User pools have a variety of features that you can turn on and off. For example, you can turn on multi-factor authentication (MFA) and turn off sign-in with third-party identity providers (IdPs). Some changes require you to switch your feature plan. The following characteristics of your user pool determine the cost that AWS bills you monthly for usage.

- The features that you choose
- The requests per second that your application makes to the user pools API
- The number of users with authentication, update, or query activity in a month, also called [monthly active users](#) or MAUs
- The number of monthly active users from third-party SAML 2.0 or OpenID Connect (OIDC) IdPs
- The number of app clients and user pools that do client-credentials grants for machine-to-machine authorization

For the most current information about user pool pricing, see [Amazon Cognito pricing](#).

Feature-plan selections apply to one user pool. Different user pools in the same AWS account can have different plan selections. You can't apply separate feature plans to app clients within a user pool. The default plan selection for new user pools is Essentials.

You can switch between feature plans at any time to fit the requirements of your applications. Some changes between plans require that you turn off active features. For more information, see [Turning off features to change feature plans](#).

User pool feature plans

Lite

Lite is a low-cost feature plan for user pools with lower numbers of monthly active users. This plan is sufficient for user directories with basic authentication features. It includes sign-in features and the classic hosted UI, a slimmer, less-customizable version of managed login. Many newer features, like access-token customization and passkey authentication, aren't included in the Lite plan.

Essentials

Essentials has all of the latest user pool authentication features. This plan adds new options to your applications, whether your login pages are managed login or custom-built. Essentials has advanced authentication features like [choice-based sign-in](#) and [email MFA](#).

Plus

Plus includes everything in the Essentials plan and adds advanced security features that protect your users. Monitor user sign-in, sign-up, and password-management requests for indicators of compromise. For example, user pools can detect whether users are signing in from an unexpected location or using a password that's been part of a public breach.

User pools with the Plus plan generate logs of user activity details and risk evaluations. You can apply your own usage and security analysis to these logs when you export them to external services.

Note

Previously, some user pool features were included in an *advanced security features* pricing structure. The features that were included in this structure are now under either the Essentials or Plus plan.

Topics

- [Select a feature plan](#)
- [Features by plan](#)
- [Essentials plan features](#)
- [Plus plan features](#)
- [Turning off features to change feature plans](#)

Select a feature plan

AWS Management Console

To choose a feature plan

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Select the **Settings** menu and review the **Feature plans** tab.
5. Review the features available to you in the Lite, Essentials, and Plus plans.
6. To change your plan, select **Switch to Essentials**, or **Switch to Plus**. To switch to the **Lite** plan, choose **Other plans**, then **Compare with Lite**.
7. On the next screen, review your choice and select **Confirm**.

CLI/API/SDK

The [CreateUserPool](#) and [UpdateUserPool](#) operations set your feature plan in the `UserPoolTier` parameter. When you don't specify a value for `UserPoolTier`, your user pool defaults to **Essentials**. If you set `AdvancedSecurityMode` to `AUDIT` or `ENFORCED`, your user pool tier must be `PLUS` and default to `PLUS` when not specified.

See [Examples in CreateUserPool](#) for syntax. See [See Also in CreateUserPool](#) for links to this function in of AWS SDKs for a variety of programming languages.

```
"UserPoolTier": "PLUS"
```

In the AWS CLI, this option is `--user-pool-tier` argument.

```
--user-pool-tier PLUS
```

See [create-user-pool](#) and [update-user-pool](#) in the AWS CLI command reference for more information.

Features by plan

Features and plans in user pools

Feature	Description	Feature plan
Protect against unsafe passwords	Check plaintext passwords for indicators or compromise at runtime	Plus
Protect against malicious sign-in attempts	Check session properties for indicators of compromise at runtime	Plus
Log and analyze user activity	Generate logs of user authentication session properties and risk scores	Plus
Export user activity logs	Push user session and risk logs to an external AWS service	Plus
Customize managed login pages with a visual editor	Use a visual editor in the Amazon Cognito console to apply branding and style to your managed login pages	Essentials + Plus
MFA with email one-time codes	Request or require local users to provide an additional email message sign-in factor after username authentication	Essentials + Plus
Customize access token scopes and claims at runtime	Use a Lambda trigger to extend the authorization	Essentials + Plus

Feature	Description	Feature plan
	capabilities of user pool access tokens	
Passwordless sign-in with one-time codes	Permit users to receive a one-time password by email or SMS as their first authentication factor	Essentials + Plus
Passkey sign-in with hardware or software FIDO2 authenticators	Permit users to use a cryptographic key stored on a FIDO2 authenticator as their first authentication factor	Essentials + Plus
Sign-up and sign-in		Lite + Essentials + Plus
User groups		Lite + Essentials + Plus
Sign-in with social, SAML, and OIDC providers	Provide users with the options to sign in directly or with their preferred provider.	Lite + Essentials + Plus
OAuth 2.0 and OIDC authorization server		Lite + Essentials + Plus
Managed login pages		Lite + Essentials + Plus
Password, custom, refresh-token, and SRP authentication	Prompt users for a username and password in your application.	Lite + Essentials + Plus
Machine-to-machine (M2M) with client credentials		Lite + Essentials + Plus
API authorization with resource servers		Lite + Essentials + Plus
User import		Lite + Essentials + Plus

Feature	Description	Feature plan
MFA with authenticator apps and SMS one-time codes	Request or require local users to provide an additional SMS message or authenticator app sign-in factor after username authentication	Lite + Essentials + Plus
Customize ID token scopes and claims at runtime	Use a Lambda trigger to extend the authentication capabilities of user pool identity (ID) tokens	Lite + Essentials + Plus
Custom runtime actions with Lambda triggers	Customize the sign-in process at runtime with Lambda functions that perform external actions and influence authentication	Lite + Essentials + Plus
Customize managed login pages with CSS	Download a CSS template and change some styles in your managed login pages	Lite + Essentials + Plus

Essentials plan features

The Essentials feature plan has most of the best and latest features of Amazon Cognito user pools. When you switch from the Lite to the Essentials plan, you get new features for your managed login pages, multi-factor authentication with email-message one-time passwords, an enhanced password policy, and custom access tokens. To stay up-to-date with new user pool features, choose the Essentials plan for your user pools.

The sections that follows present a brief overview of the features that you can add to your application with the Essentials plan. For detailed information, see the following pages.

Additional resources

- Access token customization: [Pre token generation Lambda trigger](#)
- Email MFA: [SMS and email message MFA](#)

- Password history: [Passwords, account recovery, and password policies](#)
- Enhanced UI: [Apply branding to managed login pages](#)

Topics

- [Access token customization](#)
- [Email MFA](#)
- [Password reuse prevention](#)
- [Managed login hosted sign-in and authorization server](#)
- [Choice-based authentication](#)

Access token customization

User pool [access tokens](#) grant permissions to applications: to [access an API](#), to retrieve user attributes from the [userInfo endpoint](#), or to establish [group membership](#) for an external system. In advanced scenarios, you might want to add to the default access-token data from the user pool directory with additional temporary parameters that your application determines at runtime. For example, you might want to verify a user's API permissions with [Amazon Verified Permissions](#) and adjust the scopes in the access token accordingly.

The Essentials plan adds to the existing functions of a [pre token generation trigger](#). With lower-tier plans, you can customize ID tokens with additional claims, roles, and group membership. With Essentials, you can additionally customize access tokens with claims, roles, group membership, and OAuth scopes. Access token customization isn't available to machine-to-machine (M2M) [client credentials grants](#).

To customize access tokens

1. Select the Essentials or Plus feature plan.
2. Create a Lambda function for your trigger. To use our example function, [configure it for Node.js](#).
3. Populate your Lambda function with our [example code](#) or compose your own. Your function must process a request object from Amazon Cognito and return the changes that you want to include.
4. Assign your new function as a [version 2](#) pre token generation trigger.

Learn more

- [Customizing the access token](#)
- [How to customize access tokens in Amazon Cognito user pools](#)

Email MFA

Amazon Cognito user pools can be configured to use email as the second factor in multi-factor authentication (MFA). With email MFA, Amazon Cognito can send users an email with a verification code that they must enter to complete the authentication process. This adds an important extra layer of security to the user login flow. To enable email-based MFA, the user pool must be configured to use the [Amazon SES email-sending configuration](#) instead of the default email configuration.

When your user selects MFA by email message, Amazon Cognito will send a one-time verification code to the user's registered email address whenever they attempt to sign in. The user must then provide this code back to your user pool to complete the authentication flow and gain access. This ensures that even if a user's username and password are compromised, they must provide an additional factor—the emailed code—before they can access your application resources.

For more information, see [SMS and email message MFA](#). The following is an overview of how to set up your user pool and users for email MFA.

To set up email MFA in the Amazon Cognito console

1. Select the Essentials or Plus feature plan.
2. In the **Sign-in** menu of your user pool, edit **Multi-factor authentication**.
3. Choose the level of **MFA enforcement** that you want to set up. With **Require MFA**, users in the API automatically receive a challenge to set up, confirm, and sign in with MFA. In user pools that require MFA, managed login prompts them to choose and set up an MFA factor. With **Optional MFA**, your application must offer users the option to set up MFA and set the user's preference for email MFA.
4. Under **MFA methods**, select **Email message** as one of the options.

Learn more

- [SMS and email message MFA](#)

Password reuse prevention

By default, a Amazon Cognito user pools password policy sets password length and character-type requirements, and temporary-password expiration. The Essentials plan adds the capability to enforce password history. When a user attempts to reset their password, your user pool can prevent them from setting it to a previous password. For more information about configuring the password policy, see [Adding user pool password requirements](#). The following is an overview of how to set up your user pool with a password-history policy.

To set up password history in the Amazon Cognito console

1. Select the Essentials or Plus feature plan.
2. In the **Authentication methods** menu of your user pool, locate **Password policy** and select **Edit**.
3. Configure other available options and set a value for **Prevent use of previous passwords**.

Learn more

- [Passwords, account recovery, and password policies](#)

Managed login hosted sign-in and authorization server

Amazon Cognito user pools have optional webpages that support the following functions: an OpenID Connect (OIDC) IdP, a service provider or relying party to third-party IdPs, and public user-interactive pages for sign-up and sign-in. These pages are collectively called *managed login*. When you choose a domain for your user pool, Amazon Cognito automatically activates these pages. Where the Lite plan has the hosted UI, the Essentials plan opens up this advanced version of sign-up and sign-in pages.

Managed login pages have a clean, up-to-date interface with more features and options for customizing your branding and styles. The Essentials plan is the lowest plan level that unlocks access to managed login.

To set up managed login in the Amazon Cognito console

1. From the **Settings** menu, select the Essentials or Plus feature plan.
2. From the **Domain** menu, [Assign a domain](#) to your user pool and select a **Branding version** of **Managed login**.

3. From the **Managed login** menu, under **Styles** tab, choose **Create a style** and assign the style to an app client, or create a new app client.

Learn more

- [User pool managed login](#)

Choice-based authentication

The Essentials tier introduces a new *authentication flow* for authentication operations in the enhanced UI and SDK-based API operations. This flow is *choice-based authentication*. Choice-based authentication is a method where your users' authentication starts not with an application-side declaration of a sign-in method, but a query of possible sign-in methods followed by a choice. You can configure your user pool to support choice-based authentication and unlock username-password, passwordless, and passkey authentication. In the API, this is the USER_AUTH flow.

To set up choice-based authentication in the Amazon Cognito console

1. Select the Essentials or Plus feature plan.
2. In the **Sign-in** menu of your user pool, edit **Options for choice-based sign-in**. Select and configure the authentication methods you want to enable in choice-based authentication.
3. In the **Authentication methods** menu of your user pool, edit the configuration of sign-in operations.

Learn more

- [Authentication with Amazon Cognito user pools](#)

Plus plan features

The Plus feature plan has advanced security features for Amazon Cognito user pools. These features log and analyze user context at runtime for potential security issues in devices, locations, request data, and passwords. They then mitigate potential risks with automatic responses that block or add security safeguards to user accounts. You can also export your security logs to Amazon S3, Amazon Data Firehose, or Amazon CloudWatch Logs for further analysis.

When you switch from the Essentials to the Plus plan, you get all the features in Essentials and the additional features that follow. These include the threat protection set of security options also known as *advanced security features*. To configure your user pools to automatically adapt to threats in your authentication front end, choose the Plus plan for your user pools.

The sections that follows present a brief overview of the features that you can add to your application with the Plus plan. For detailed information, see the following pages.

Additional resources

- Adaptive authentication: [Working with adaptive authentication](#)
- Compromised credentials: [Working with compromised-credentials detection](#)
- Log export: [Exporting logs from Amazon Cognito user pools](#)

Topics

- [Threat protection: adaptive authentication](#)
- [Threat protection: compromised-credentials detection](#)
- [Threat protection: user activity logging](#)

Threat protection: adaptive authentication

The Plus plan includes an *adaptive authentication* feature. When you activate this feature, your user pool makes a risk assessment of every user authentication session. From the resulting risk ratings, you can block authentication or push MFA for users who sign in with a risk level above a threshold that you determine. With adaptive authentication, your user pool and application automatically block or set up MFA for users whose accounts you suspect are being attacked. You can also provide feedback on the risk ratings from your user pool to adjust future ratings.

To set up adaptive authentication in the Amazon Cognito console

1. Select the Plus feature plan.
2. From the **Threat protection** menu of your user pool, edit **Standard and custom authentication** under **Threat protection**.
3. Set the **enforcement mode** for standard or custom authentication to **Full-function**.
4. Under **Adaptive authentication**, configure automatic risk responses for different levels of risk.

Learn more

- [Working with adaptive authentication](#)
- [Collecting data for threat protection in applications](#)

Threat protection: compromised-credentials detection

The Plus plan includes a *compromised-credentials detection* feature. This feature guards against the use of insecure passwords and the threat of unintended application access that this practice creates. When you permit your users to sign in with username and password, they might reuse a password that they've used elsewhere. That password might have been leaked, or just be commonly guessed. With compromised-credentials detection, your user pool reads the passwords your users submit and compares them to password databases. If the operation results in a decision that the password is likely compromised, you can configure your user pool to block sign-in and then initiate a password reset for the user in your application.

Compromised-credentials detection can react to insecure passwords when new users sign up, when existing users sign in, and when users attempt to reset their passwords. With this feature, your user pool can prevent or warn about sign-in with insecure passwords wherever users enter them.

To set up compromised-credentials detection in the Amazon Cognito console

1. Select the Plus feature plan.
2. From the **Threat protection** menu of your user pool, edit **Standard and custom authentication** under **Threat protection**.
3. Set the **enforcement mode** for standard or custom authentication to **Full-function**.
4. Under **Compromised credentials**, configure the types of authentication operations that you want to check, and the automated response that you want from your user pool.

Learn more

- [Working with compromised-credentials detection](#)

Threat protection: user activity logging

The Plus plan adds a logging feature that gives security analysis and details of user authentication attempts. You can see risk assessments, user IP addresses, user agents, and other information

about the device that connected to your application. You can act on this information with the built-in threat protection features, or you can analyze your logs in your own systems and take appropriate action. You can export the logs from threat protection to Amazon S3, CloudWatch Logs, or Amazon DynamoDB.

To set up user activity logging in the Amazon Cognito console

1. Select the Plus feature plan.
2. From the **Threat protection** menu of your user pool, edit **Standard and custom authentication** under **Threat protection**.
3. Set the **enforcement mode** for standard or custom authentication to **Audit-only**. This is the minimum setting for logs. You can also activate it in **Full-function** mode and configure other threat protection features.
4. To export your logs to another AWS service for third-party analysis, go to the **Log streaming** menu of your user pool and set up an export destination.

Learn more

- [Exporting user authentication events](#)
- [Exporting logs from Amazon Cognito user pools](#)

Turning off features to change feature plans

Feature plans add configuration options to your user pool. You can configure and use these features only when the related feature plan is active. For example, you can configure access token customization in the Plus and Essentials plans, but not in the Lite plan. To deactivate these features, you must deactivate each active component. The **Switch to** option in the **Settings** menu in the Amazon Cognito console notifies you of the features you must deactivate before you can change your feature plan. With this chapter, you can learn the changes that deactivation makes to your user pool configuration, and how to turn off these features individually.

Access token customization

To switch to a plan that doesn't include access token customization, you must remove the [pre token generation Lambda trigger](#) from your user pool. To add a new pre token generation trigger without access token customization, assign a new function to the trigger and configure it for V1_0 events. These version one trigger events can process changes to ID tokens only.

To manually deactivate access token customization, remove your pre token generation trigger and add a new version one trigger.

Threat protection

To switch to a plan without threat protection, deactivate all features from the **Threat protection** menu of your user pool.

Log export

To switch to a plan without log export, deactivate it from the **Log streaming** menu of your user pool. Your user pool no longer generates local or exported user-activity logs. You can also send a [SetLogDeliveryConfiguration](#) API request that removes any configuration with an EventSource value of UserActivity.

Email MFA

To switch to a plan without email MFA, go to the **Sign-in** menu of your user pool. Edit **Multi-factor authentication** and deselect **Email message** as one of the available **MFA methods**.

Authentication with Amazon Cognito user pools

Amazon Cognito includes several methods to authenticate your users. All user pools, whether you have a domain or not, can authenticate users in the user pools API. If you add a domain to your user pool, you can use the [user pool endpoints](#). The user pools API supports a variety of authorization models and request flows for API requests.

To verify the identity of users, Amazon Cognito supports authentication flows that incorporate challenge types in addition to passwords like email and SMS message one-time passwords and passkeys.

Topics

- [Implement authentication flows](#)
- [Things to know about authentication with user pools](#)
- [An example authentication session](#)
- [Configure authentication methods for managed login](#)
- [Manage authentication methods in AWS SDKs](#)
- [Authentication flows](#)
- [Authorization models for API and SDK authentication](#)

- [Application resources for user pool authentication](#)

Implement authentication flows

Whether you're implementing [managed login](#) or a [custom-built application front end](#) with an AWS SDK for authentication, you must configure your app client for the types of authentication that you want to implement. The following information describes setup for authentication flows in your [app clients](#) and your application.

App client supported flows

You can configure supported flows for your app clients in the Amazon Cognito console or with the API in an AWS SDK. After you configure your app client to support these flows, you can deploy them in your application.

The following procedure configures available authentication flows for an app client with the Amazon Cognito console.

To configure an app client for authentication flows (console)

1. Sign in to AWS and navigate to the [Amazon Cognito user pools console](#). Choose a user pool or create a new one.
2. In your user pool configuration, select the **App clients** menu. Choose an app client or create a new one.
3. Under **App client information**, select **Edit**.
4. Under **App client flows**, choose the authentication flows that you want to support.

To configure an app client for authentication flows (API/SDK)

To configure available authentication flows for an app client with the Amazon Cognito API, set the value of `ExplicitAuthFlows` in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) request. The following is an example that provisions secure remote password (SRP) and choice-based authentication to a client.

```
"ExplicitAuthFlows": [  
    "ALLOW_USER_AUTH",  
    "ALLOW_USER_SRSP_AUTH  
]
```

When you configure app client supported flows, you can specify the following options and API values.

App client flow support

Authentication flow	Compatibility	Console	API
<u>Choice-based authentication</u>	Server-side, client-side	Select an authentication type at sign-in	ALLOW_USE_R_AUTH
<u>Sign-in with persistent passwords</u>	Client-side	Sign in with username and password	ALLOW_USE_R_PASSWORD_AUTH
<u>Sign-in with persistent passwords and secure payload</u>	Server-side, client-side	Sign in with secure remote password (SRP)	ALLOW_USE_R_SRP_AUTH
<u>Refresh tokens</u>	Server-side, client-side	Get new user tokens from existing authenticated sessions	ALLOW_REFRESH_TOKEN_AUTH
<u>Server-side authentication</u>	Server-side	Sign in with server-side administrative credentials	ALLOW_ADMIN_USER_PASSWORD_AUTH
<u>Custom authentication</u>	Server-side and client-side custom-built applications. Not compatible with managed login.	Sign in with custom authentication flows from Lambda triggers	ALLOW_CUSTOM_AUTH

Implement flows in your application

Managed login automatically makes your configured authentication options available in your sign-pages. In custom-built applications, start authentication with a declaration of the initial flow.

- To choose from a list of flow options for a user, declare [choice-based authentication](#) with the USER_AUTH flow. This flow has available authentication methods that aren't available in client-based authentication flows, for example [passkey](#) and [passwordless](#) authentication.
- To choose your authentication flow up front, declare [client-based authentication](#) with any other flow that's available in your app client.

When you sign users in, the body of your [InitiateAuth](#) or [AdminInitiateAuth](#) request must include an AuthFlow parameter.

Choice-based authentication:

```
"AuthFlow": "USER_AUTH"
```

Client-based authentication with SRP:

```
"AuthFlow": "USER_SRP_AUTH"
```

Things to know about authentication with user pools

Consider the following information in the design of your authentication model with Amazon Cognito user pools.

Authentication flows in managed login and the hosted UI

[Managed login](#) and the classic hosted UI have different options for authentication. You can only do passwordless and passkey authentication in managed login.

Custom authentication flows only available in AWS SDK authentication

You can't do *custom authentication flows*, or [custom authentication with Lambda triggers](#), with managed login or the classic hosted UI. Custom authentication is available in [authentication with AWS SDKs](#).

Managed login for external identity provider (IdP) sign-in

You can't sign users in through [third-party IdPs](#) in [authentication with AWS SDKs](#). You must implement managed login or the classic hosted UI, redirect to IdPs, and then process the resulting authentication object with OIDC libraries in your application. For more information about managed login, see [User pool managed login](#).

Passwordless authentication effect on other user features

Activation of [passwordless sign-in](#) in your user pool and app client has an effect on user creation and migration. When passwordless sign-in is active:

1. Administrators can create users without passwords. The default invitation message template changes to no longer include the {###} password placeholder. For more information, see [Creating user accounts as administrator](#).
2. For SDK-based [SignUp](#) operations, users aren't required to supply a password when they sign up. Managed login and the hosted UI require a password in the sign-up page, even if passwordless authentication is permitted. For more information, see [Signing up and confirming user accounts](#).
3. Users imported from a CSV file can sign in immediately with passwordless options, without a password reset, if their attributes include an email address or phone number for an available passwordless sign-in option. For more information, see [Importing users into user pools from a CSV file](#).
4. Passwordless authentication doesn't invoke the [user migration Lambda trigger](#).

Passkey relying party URLs can't be on the public suffix list

You can use domain names that you own, like `www.example.com`, as the relying party (RP) ID in your passkey configuration. This configuration is intended to support custom-built applications that run on domains that you own. The [public suffix list](#), or PSL, contains protected high-level domains. Amazon Cognito returns an error when you attempt to set your RP URL to a domain on the PSL.

Topics

- [Authentication session flow duration](#)
- [Lockout behavior for failed sign-in attempts](#)

Authentication session flow duration

Depending on the features of your user pool, you can end up responding to several challenges to `InitiateAuth` and `RespondToAuthChallenge` before your app retrieves tokens from Amazon Cognito. Amazon Cognito includes a session string in the response to each request. To combine your API requests into an authentication flow, include the session string from the response to the previous request in each subsequent request. By default, your users have three minutes to

complete each challenge before the session string expires. To adjust this period, change your app client **Authentication flow session duration**. The following procedure describes how to change this setting in your app client configuration.

 **Note**

Authentication flow session duration settings apply to authentication with the Amazon Cognito user pools API. Managed login sets session duration to 3 minutes for multi-factor authentication and 8 minutes for password-reset codes.

Amazon Cognito console

To configure app client authentication flow session duration (AWS Management Console)

1. From the **App integration** tab in your user pool, select the name of your app client from the **App clients and analytics** container.
2. Choose **Edit** in the **App client information** container.
3. Change the value of **Authentication flow session duration** to the validity duration that you want, in minutes, for SMS MFA codes. This also changes the amount of time that any user has to complete any authentication challenge in your app client.
4. Choose **Save changes**.

User pools API

To configure app client authentication flow session duration (Amazon Cognito API)

1. Prepare an `UpdateUserPoolClient` request with your existing user pool settings from a `DescribeUserPoolClient` request. Your `UpdateUserPoolClient` request must include all existing app client properties.
2. Change the value of `AuthSessionValidity` to the validity duration that you want, in minutes, for SMS MFA codes. This also changes the amount of time that any user has to complete any authentication challenge in your app client.

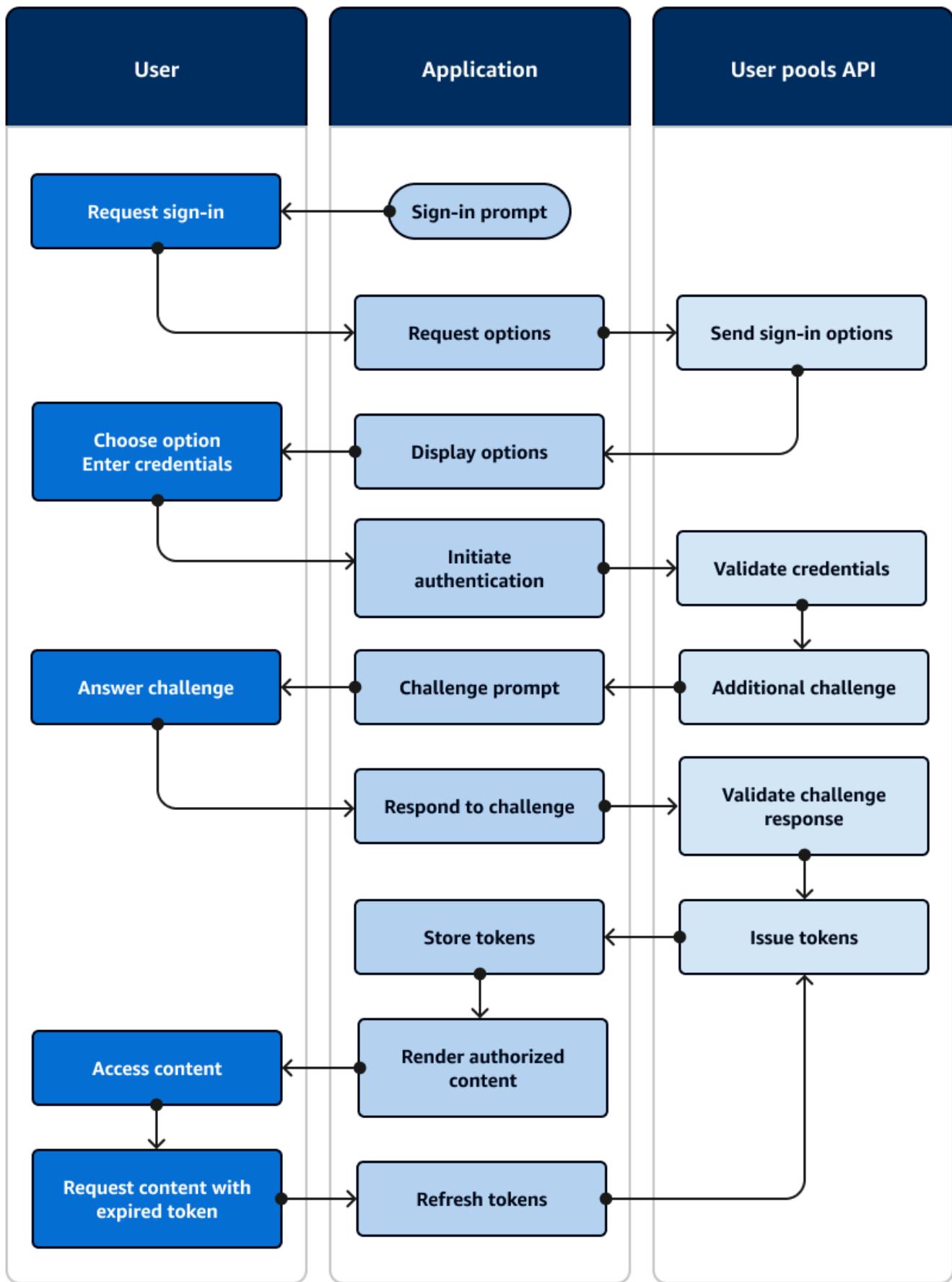
For more information about app clients, see [Application-specific settings with app clients](#).

Lockout behavior for failed sign-in attempts

After five failed unauthenticated or IAM-authenticated sign-in attempts with a password, Amazon Cognito locks out your user for one second. The lockout duration then doubles after each additional one failed attempt, up to a maximum of approximately 15 minutes. Attempts made during a lockout period generate a `Password attempts exceeded` exception, and don't affect the duration of subsequent lockout periods. For a cumulative number of failed sign-in attempts n , not including `Password attempts exceeded` exceptions, Amazon Cognito locks out your user for $2^{(n-5)}$ seconds. To reset the lockout to its $n=0$ initial state, your user must either sign in successfully after a lockout period expires, or not initiate any sign-in attempts for 15 consecutive minutes at any time after a lockout. This behavior is subject to change. This behavior doesn't apply to custom challenges unless they also perform password-based authentication.

An example authentication session

The following diagram and step-by-step guide illustrate a typical scenario where a user signs in to an application. The example application presents a user with several sign-in options. They select one by entering their credentials, provide an additional authentication factor, and sign in.



Picture an application with a sign-in page where users can sign in with a username and password, request a one-time code in an email message, or choose a fingerprint option.

1. **Sign-in prompt:** Your application shows a home screen with a *Log in* button.
2. **Request sign-in:** The user selects *Log in*. From a cookie or a cache, your application retrieves their username, or prompts them to enter it.
3. **Request options:** Your application requests the user's sign-in options with an `InitiateAuth` API request with the `USER_AUTH` flow, requesting the available sign-in methods for the user.
4. **Send sign-in options:** Amazon Cognito responds with `PASSWORD`, `EMAIL OTP`, and `WEB AUTHN`. The response includes a session identifier for you to replay back in the next response.
5. **Display options:** Your application shows UI elements for the user to enter their username and password, get a one-time code, or scan their fingerprint.
6. **Choose option/Enter credentials:** The user enters their username and password.
7. **Initiate authentication:** Your application provides the user's sign-in information with a `RespondToAuthChallenge` API request that confirms username-password sign-in and provides the username and the password.
8. **Validate credentials:** Amazon Cognito confirms the user's credentials.
9. **Additional challenge:** The user has multi-factor authentication configured with an authenticator app. Amazon Cognito returns a `SOFTWARE_TOKEN_MFA` challenge.
10. **Challenge prompt:** Your application displays a form requesting a time-based one-time password (TOTP) from the user's authenticator app.
11. **Answer challenge:** The user submits the TOTP.
12. **Respond to challenge:** In another `RespondToAuthChallenge` request, your application provides the user's TOTP.
13. **Validate challenge response:** Amazon Cognito confirms the user's code and determines that your user pool is configured to issue no additional challenges to the current user.
14. **Issue tokens:** Amazon Cognito returns ID, access, and refresh JSON web tokens (JWTs). The user's initial authentication is complete.
15. **Store tokens:** Your application caches the user's tokens so that it can reference user data, authorize access to resources, and update tokens when they expire.
16. **Render authorized content:** Your application makes a determination of the user's access to resources based on their identity and roles, and delivers application content.
17. **Access content:** The user is signed in and begins using the application.

18Request content with expired token: Later, the user requests a resource that requires authorization. The user's cached token has expired.

19Refresh tokens: Your application makes an `InitiateAuth` request with the user's saved refresh token.

20Issue tokens: Amazon Cognito returns new ID and access JWTs. The user's session is securely refreshed without additional prompts for credentials.

You can use [AWS Lambda triggers](#) to customize the way users authenticate. These triggers issue and verify their own challenges as part of the authentication flow.

You can also use the admin authentication flow for secure backend servers. You can use the [user migration authentication flow](#) to make user migration possible without the requirement that your users to reset their passwords.

Configure authentication methods for managed login

You can invoke [managed login pages](#) when you want users to sign in, sign out, or reset their password. In this model, your application imports OIDC libraries to process browser-based authentication attempts with user pool managed login pages. The forms of authentication that are available to your users are dependent on the configuration of your user pool and your app client. Implement the `ALLOW_USER_AUTH` flow in your app client, and Amazon Cognito prompts users to select a sign-in method from the available options. Implement `ALLOW_USER_PASSWORD_AUTH` and assign a SAML provider, and your login pages prompt users with the option to enter their username and password or to connect with their IdP.

The Amazon Cognito user pools console can get you started with setting up managed login authentication for your application. When you create a new user pool, specify the platform you're developing for and the console gives you examples for implementation of OIDC and OAuth libraries with starter code to implement sign-in and sign-out flows. You can build managed login with many OIDC relying-party implementations. We recommend that you work with [certified OIDC relying party libraries](#) where possible. For more information, see [Getting started with user pools](#).

User pool settings for managed login

You might want to allow sign in with multiple providers for your application, or you might want to use Amazon Cognito as an independent user directory. You might also want to collect user attributes, set up and prompt for MFA, or require email addresses as usernames. You can't directly

edit the fields in managed login and the hosted UI. Instead, the configuration of your user pool automatically sets the handling of managed-login authentication flows.

The following user pool configuration items determine the authentication methods that Amazon Cognito presents to users in managed login and the hosted UI.

User pool options (Sign-in menu)

The following options are in the **Sign-in** menu of a user pool in the Amazon Cognito console.

Cognito user pool sign-in options

Has options for usernames. Your managed login and hosted UI pages only accept usernames in the formats that you select. When you, for example, set up a user pool with **Email** as the only sign-in option, your managed login pages only accept usernames in an email format.

Required attributes

When you set an attribute as required in your user pool, managed login prompts users for a value for that attribute when they sign up.

Options for choice-based sign-in

Has settings for authentication methods in [Choice-based authentication](#). Here, you can turn on or off authentication methods like [passkey](#) and [passwordless](#). These methods are only available to user pools with [managed login domains](#) and [feature plans](#) above the **Lite** tier.

Multi-factor authentication

Managed login and the hosted UI handle registration and authentication operations for [MFA](#). When MFA is required in your user pool, your sign-in pages automatically prompt users to set up their additional factor. They also prompt users who have an MFA configuration to complete authentication with an MFA code. When MFA is off or optional in your user pool, your sign-in pages don't prompt to set up MFA.

User account recovery

The self-service [account recovery](#) setting of your user pool determines whether your sign-in pages display a link where users can reset their password.

User pool options (Domain menu)

The following options are in the **Domain** menu of a user pool in the Amazon Cognito console.

Domain

Your choice of a user pool domain sets the path for the link that users open when you invoke their browsers for authentication.

Branding version

Your choice of a branding version determines whether your user pool domain displays managed login or the hosted UI.

User pool options (Social and external providers menu)

The following option is in the **Social and external providers** menu of a user pool in the Amazon Cognito console.

Providers

The identity providers (IdPs) that you add to your user pool can be left active or inactive for each app client in the user pool.

App client options

The following options are in the **App clients** menu of a user pool in the Amazon Cognito console. To review these options, select an app client from the list.

Quick setup guide

The quick setup guide has code examples for a variety of developer environments. They contain the libraries necessary to integrate managed login authentication with your application.

App client information

Edit this configuration to set assigned IdPs for the application that's represented by the current app client. On the managed login pages, Amazon Cognito displays choices for users. These choices are determined from the assigned methods and IdP. For example, if you assign a SAML 2.0 IdP named MySAML and local user pool login, your managed login pages display authentication-method prompts and a button for MySAML.

Authentication settings

Edit this configuration to set authentication methods for your application. On the managed login pages, Amazon Cognito displays choices for users. These choices are determined from the

availability of the user pool as an IdP, and from the methods that you assign. For example, if you assign choice-based ALLOW_USER_AUTH authentication, your managed login pages display available choices like entering an email address and signing in with a passkey. Managed login pages also render buttons for the assigned IdPs.

Login pages

Set the visual effect of your managed login or hosted UI user-interactive pages with the options available in this tab. For more information, see [Apply branding to managed login pages](#).

Manage authentication methods in AWS SDKs

Users in Amazon Cognito user pools can sign in with a variety of initial sign-in options, or *factors*. For some factors, users can follow up with multi-factor authentication (MFA). These first factors include username and password, one-time password, passkey, and custom authentication. For more information, see [Authentication flows](#). When your application has built-in UI components and imports an AWS SDK module, you must build application logic for authentication. You must choose one of two primary methods and from that method, the authentication mechanisms that you want to implement.

You can implement *client-based authentication* where your application, or client, declares the type of authentication up front. Your other option is *choice-based authentication*, where your app collects a username and requests the available authentication types for users. You can implement these models together in the same application or split between app clients, according to your requirements. Each method has features that are unique to it, for example custom authentication in client-based and passwordless authentication in choice-based.

In custom-built applications that perform authentication with AWS SDK implementation of the users pools API, you must structure your API requests to align with user pool configuration, app client configuration, and client-side preferences. An `InitiateAuth` session that begins with an `AuthFlow` of `USER_AUTH` begins choice-based authentication. Amazon Cognito responds to your API with a challenge of either a preferred authentication method or a list of choices. A session that begins with `AuthFlow` of `CUSTOM_AUTH` goes right into custom authentication with Lambda triggers.

Some authentication methods are fixed to one of the two flow types, and some methods are available in both.

Topics

- [Choice-based authentication](#)
- [Client-based authentication](#)

Choice-based authentication

Your application can request the following authentication methods in choice-based authentication.

1. EMAIL_OTP and SMS_OTP

[Passwordless sign-in with one-time passwords](#)

2. WEB_AUTHN

[Passkey sign-in](#)

3. PASSWORD

[Sign-in with persistent passwords](#)

[Sign-in with persistent passwords and secure payload](#)

[MFA after sign-in](#)

To review these options in their API context, see ChallengeName in [RespondToAuthChallenge](#).

Choice-based sign-in issues a challenge in response to your initial request. This challenge either verifies that a requested option is available, or provides a list of available choices. Your application can display these choices to users, who then enter credentials for their preferred sign-in method and proceed with authentication in challenge responses.

You have the following choice-based options in your authentication flow. All requests of this type require that your app first collect a username or retrieve it from a cache.

1. Request options with AuthParameters of USERNAME only. Amazon Cognito returns a SELECT_CHALLENGE challenge. From there, your application can prompt the user to select a challenge and return this response to your user pool.
2. Request a preferred challenge with AuthParameters of PREFERRED_CHALLENGE. If your user, user pool, and app client are all configured for the preferred challenge, Amazon Cognito responds with that challenge. If the preferred challenge isn't available, Amazon Cognito responds with SELECT_CHALLENGE and a list of available challenges.

3. Sign users in first, then request their choice-based authentication options. A [GetUserAuthFactors](#) request with the access token of a signed-in user returns their available choice-based authentication factors and their MFA settings. With this option, a user can sign in with username and password first, then activate a different form of authentication. You can also use this operation to check additional options for a user who has signed in with a preferred challenge.

Client-based authentication

Client-based authentication supports the following authentication flows.

1. USER_PASSWORD_AUTH and ADMIN_USER_PASSWORD_AUTH

[Sign-in with persistent passwords](#)

[MFA after sign-in](#)

2. USER_SRP_AUTH

[Sign-in with persistent passwords and secure payload](#)

[MFA after sign-in](#)

3. REFRESH_TOKEN_AUTH

[Refresh tokens](#)

4. CUSTOM_AUTH

[Custom authentication](#)

Client-based authentication is an assumption that your application has determined how your user wants to authenticate before it begins an authentication flow. The `InitiateAuth` request declares a sign-in `AuthFlow` that directly corresponds to one of the listed options, for example `USER_SRP_AUTH`. With this declaration, the request also includes the parameters to begin authentication, for example `USERNAME`, `SECRET_HASH`, and `SRP_A`. Amazon Cognito might follow up this request with additional challenges like `PASSWORD_VERIFIER` for SRP or `SOFTWARE_TOKEN_MFA` for password sign-in with TOTP MFA.

Authentication flows

The process of authentication with Amazon Cognito user pools can best be described as a *flow* where users make an initial choice, submit credentials, and respond to additional challenges. When you implement managed login authentication in your application, Amazon Cognito manages the flow of these prompts and challenges. When you implement flows with an AWS SDK in your application back end, you must construct the logic of requests, prompt users for input, and respond to challenges.

As an application administrator, your user characteristics, security requirements, and authorization model help determine how you want to permit users to sign in. Ask yourself the following questions.

- Do I want to permit users to sign in with credentials from [other identity providers \(IdPs\)](#)?
- Is a [username and password](#) enough proof of identity?
- Could my authentication requests for username-password authentication be intercepted? Do I want my application to transmit passwords, or to [negotiate authentication using hashes and salts](#)?
- Do I want to permit users to skip password entry and [receive a one-time password](#) that signs them in?
- Do I want to permit users to sign in with a [thumbprint, face, or a hardware security key](#)?
- When do I want to require [multi-factor authentication \(MFA\)](#), if at all?
- Do I want to [persist user sessions without re-prompting for credentials](#)?
- Do I want to [extend my authorization model](#) beyond the built-in capabilities of Amazon Cognito?

When you have the answers to these questions, you can learn how to activate the relevant features and implement them in the authentication requests that your application makes.

After you set up sign-in flows for a user, you can check their current status for MFA and [choice-based](#) authentication factors with requests to the [GetUserAuthFactors](#) API operation. This operation requires authorization with the access token of a signed-in user. It returns user authentication factors and MFA settings.

Topics

- [Sign-in with third-party IdPs](#)

- [Sign-in with persistent passwords](#)
- [Sign-in with persistent passwords and secure payload](#)
- [Passwordless sign-in with one-time passwords](#)
- [Passkey sign-in](#)
- [MFA after sign-in](#)
- [Refresh tokens](#)
- [Custom authentication](#)
- [User migration authentication flow](#)

Sign-in with third-party IdPs

Amazon Cognito user pools serve as an intermediate broker of authentication sessions between IdPs like Sign in with Apple, Login with Amazon, and OpenID Connect (OIDC) services. This process is also called *federated sign-in* or *federated authentication*. Federated authentication doesn't make use of any of the authentication flows that you can build into your app client. Instead, you assign configured user pool IdPs to your app client. Federated sign-in happens when users select their IdP in managed login or your application invokes a session with a redirect to their IdP sign-in page.

With federated sign-in, you delegate primary and MFA authentication factors to the user's IdP. Amazon Cognito doesn't add the other advanced flows in this section to a federated user unless you [link them to a local user](#). Unlinked federated users have usernames, but they are a store of mapped attribute data that's not typically used for sign-in independent of the browser-based flow.

Implementation resources

- [User pool sign-in with a third party identity provider](#)

Sign-in with persistent passwords

In Amazon Cognito user pools, every user has a username. This might be a phone number, an email address, or a chosen or administrator-provided identifier. Users of this type can sign in with their username and their password, and optionally provide MFA. User pools can perform username-password sign-in with public or IAM-authenticated API operations and SDK methods. Your application can directly send the password to your user pool for authentication. Your user pool responds with additional challenges or the JSON web tokens (JWTs) that are the result of successful authentication.

Activate password sign-in

To activate sign-in with username and password, configure your app client to permit it. In the Amazon Cognito console, navigate to the **App clients** menu under **Applications** in your user pool configuration. To permit username-password sign-in for a client-side mobile or native app, select or create an app client and choose **Sign in with username and password: ALLOW_USER_PASSWORD_AUTH**. To permit username-password sign-in for a server-side or web app, choose **Sign in with server-side administrative credentials: ALLOW_ADMIN_USER_PASSWORD_AUTH**.

In the user pools API, configure `ExplicitAuthFlows` with the required option in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) request.

```
"ExplicitAuthFlows": [
    "ALLOW_USER_PASSWORD_AUTH",
    "ALLOW_ADMIN_USER_PASSWORD_AUTH"
]
```

Choice-based sign-in with a password

To sign a user in to a client-side app with username-password authentication, configure the body of your [InitiateAuth](#) request as follows. Amazon Cognito responds with a `ChallengeName` of `PASSWORD` if the current user is eligible for username-password authentication. Otherwise, it responds with a list of available challenges. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
    "AuthFlow": "USER_AUTH",
    "AuthParameters": {
        "USERNAME" : "testuser",
        "PREFERRED_CHALLENGE" : "PASSWORD"
    },
    "ClientId": "example23456789"
}
```

To sign a user in to a server-side app with username-password authentication, configure the body of your [AdminInitiateAuth](#) request as follows. Your application must sign this request with AWS credentials. Amazon Cognito responds with a `ChallengeName` of `PASSWORD` if the current user is eligible for username-password authentication. Otherwise, it responds with a list

of available challenges. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{  
    "AuthFlow": "USER_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser",  
        "PREFERRED_CHALLENGE" : "PASSWORD"  
    },  
    "ClientId": "1example23456789"  
}
```

Client-based sign-in with a password

To sign a user in to a client-side app with username-password authentication, configure the body of your [InitiateAuth](#) request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{  
    "AuthFlow": "USER_PASSWORD_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser",  
        "PASSWORD" : "Example1234!"  
    },  
    "ClientId": "1example23456789"  
}
```

To sign a user in to a server-side app with username-password authentication, configure the body of your [AdminInitiateAuth](#) request as follows. Your application must sign this request with AWS credentials. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{  
    "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser",  
        "PASSWORD" : "Example1234!"  
    },  
    "ClientId": "1example23456789"  
}
```

Sign-in with persistent passwords and secure payload

Another form of the username-password sign-in methods in user pools is with the Secure Remote Password (SRP) protocol. This option sends proof of knowledge of a password—a password hash and a salt—that your user pool can verify. With no readable secret information in the request to Amazon Cognito, your application is the only entity that processes the passwords that users enter. SRP authentication involves mathematical calculations that are best done by an existing component that you can import in your SDK. SRP is typically implemented in client-side applications like mobile apps. For more information about the protocol, see [The Stanford SRP Homepage](#). [Wikipedia](#) also has resources and examples.

Activate SRP sign-in

To activate sign-in with username and password, configure your app client to permit it. In the Amazon Cognito console, navigate to the **App clients** menu under **Applications** in your user pool configuration. To permit username-password sign-in for a client-side mobile or native app, select or create an app client and choose **Sign in with username and password: ALLOW_USER_PASSWORD_AUTH**. To permit username-password sign-in for a server-side or web app, choose **Sign in with server-side administrative credentials: ALLOW_ADMIN_USER_PASSWORD_AUTH**.

In the user pools API, configure `ExplicitAuthFlows` with the required option in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) request.

```
"ExplicitAuthFlows": [
    "ALLOW_USER_SRP_AUTH"
]
```

Choice-based sign-in with SRP

To sign a user in to a client-side app with username-password authentication, configure the body of your [InitiateAuth](#) request as follows. Amazon Cognito responds with a `ChallengeName` of `PASSWORD_SRP` if the current user is eligible for username-password authentication. Otherwise, it responds with a list of available challenges. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
    "AuthFlow": "USER_AUTH",
    "AuthParameters": {
        "USERNAME" : "testuser",
```

```
"PREFERRED_CHALLENGE" : "PASSWORD_SRP"  
},  
"ClientId": "lexample23456789"  
}
```

To sign a user in to a server-side app with username-password authentication, configure the body of your [AdminInitiateAuth](#) request as follows. Your application must sign this request with AWS credentials. Amazon Cognito responds with a ChallengeName of PASSWORD_SRP if the current user is eligible for username-password authentication. Otherwise, it responds with a list of available challenges. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{  
    "AuthFlow": "USER_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser",  
        "PREFERRED_CHALLENGE" : "PASSWORD_SRP"  
    },  
    "ClientId": "lexample23456789"  
}
```

Client-based sign-in with SRP

SRP authentication is more common to client-side authentication than to server-side. However, you can use SRP authentication with [InitiateAuth](#) and [AdminInitiateAuth](#). To sign a user in to an application, configure the body of your InitiateAuth or AdminInitiateAuth request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

The client generates SRP_A from a generator modulo N g raised to the power of a secret random integer a .

```
{  
    "AuthFlow": "USER_PASSWORD_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser",  
        "SRP_A" : "g^a"  
    },  
    "ClientId": "lexample23456789"  
}
```

Amazon Cognito responds with a PASSWORD_VERIFIER challenge. Your client must complete SRP calculations and respond to the challenge in a [RespondToAuthChallenge](#) or [AdminRespondToAuthChallenge](#) request.

```
{  
    "ChallengeName": "PASSWORD_VERIFIER",  
    "ChallengeResponses": {  
        "PASSWORD_CLAIM_SIGNATURE" : "string",  
        "PASSWORD_CLAIM_SECRET_BLOCK" : "string",  
        "TIMESTAMP" : "string"  
    },  
    "ClientId": "1example23456789",  
    "Session": "[Session ID from the previous response]"  
}
```

Built-in authentication flow and challenges

Amazon Cognito contains built-in AuthFlow and ChallengeName values so that a standard authentication flow can validate a username and password through the Secure Remote Password (SRP) protocol. The AWS SDKs have built-in support for these flows with Amazon Cognito.

The flow starts by sending USER_SRP_AUTH as the AuthFlow to InitiateAuth. You also send USERNAME and SRP_A values in AuthParameters. If the InitiateAuth call is successful, the response has 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 signs in, Amazon Cognito issues tokens. If you activated multi-factor authentication (MFA) for the user, Amazon Cognito returns the ChallengeName of SMS_MFA. The app can provide the necessary code through another call to RespondToAuthChallenge.

Passwordless sign-in with one-time passwords

Passwords can be lost or stolen. You might want to verify only that your users have access to a verified email address, phone number, or authenticator app. The solution to this is *passwordless* sign-in. Your application can prompt users to enter their username, email address, or phone number. Amazon Cognito then generates a one-time password (OTP), a code that they must confirm. A successful code completes authentication.

When a user correctly enters a code they received in an SMS or email message as part of passwordless authentication, in addition to authenticating the user, your user pool marks the user's unverified email address or phone number attribute as verified. The user status also changes from UNCONFIRMED to CONFIRMED, regardless of whether you configured your user pool to [automatically verify](#) email addresses or phone numbers.

New options with passwordless sign-in

When you activate passwordless authentication in your user pool, it changes how some user flows work.

1. Users can sign up without a password and choose a passwordless factor when they sign in. You can also create users without passwords as an administrator.
2. Users who you [import with a CSV file](#) can sign in immediately with a passwordless factor. They aren't required to set a password before sign-in.
3. Users who don't have a password can submit [ChangePassword](#) API requests without the PreviousPassword parameter.

Automatic sign-in with OTPs

Users who sign up and confirm their user accounts with email or SMS message OTPs can automatically sign in with the passwordless factor that matches their confirmation message. In the managed login UI, users who confirm their accounts and are eligible for OTP sign-in with the confirmation-code delivery method automatically proceed through to their first sign-in after they provide the confirmation code. In your custom-built application with an AWS SDK, pass the following parameters to an [InitiateAuth](#) or [AdminInitiateAuth](#) operation.

- The Session parameter from the [ConfirmSignUp](#) API response as the Session request parameter.
- An [AuthFlow](#) of USER_AUTH.

You can pass a [PREFERRED_CHALLENGE](#) of EMAIL OTP or SMS OTP, but it's not required. The Session parameter provides proof of authentication and Amazon Cognito ignores the AuthParameters when you pass a valid session code.

The sign-in operation returns the response that indicates successful authentication, [AuthenticationResult](#), with no additional challenges if the following conditions are true.

- The Session code is still valid.
- The user is eligible for the requested PREFERRED_CHALLENGE authentication method
- The user doesn't have MFA set up. If they do have MFA, Amazon Cognito returns the appropriate challenge to complete MFA.

Activate passwordless sign-in

Console

To activate passwordless sign-in, configure your user pool to permit primary sign-in with one or more passwordless types, then configure your app client to permit the USER_AUTH flow. In the Amazon Cognito console, navigate to the **Sign-in** menu under **Authentication** in your user pool configuration. Edit **Options for choice-based sign-in** and choose **Email message one-time password** or **SMS message one-time password**. You can activate both options. Save your changes.

Navigate to the **App clients** menu and choose an app client or create a new one. Select **Edit** and choose **Select an authentication type at sign-in: ALLOW_USER_AUTH**.

API/SDK

In the user pools API, configure SignInPolicy with the appropriate passwordless options in a [CreateUserPool](#) or [UpdateUserPool](#) request.

```
"SignInPolicy": {  
    "AllowedFirstAuthFactors": [  
        "EMAIL OTP",  
        "SMS OTP"  
    ]  
}
```

Configure your app client ExplicitAuthFlows with the required option in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) request.

```
"ExplicitAuthFlows": [  
    "ALLOW_USER_AUTH"  
]
```

Sign in with passwordless

Passwordless sign-in doesn't have an AuthFlow that you can specify in [InitiateAuth](#) and [AdminInitiateAuth](#). Instead, you must declare an AuthFlow of USER_AUTH and request a sign-in option or choose your passwordless option from the response from your user pool. To sign a user in to an application, configure the body of your `InitiateAuth` or `AdminInitiateAuth` request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

In this example, we don't know which way the user wants to sign in. If we add a `PREFERRED_CHALLENGE` parameter and the preferred challenge is available to the user, Amazon Cognito responds with that challenge.

```
{  
    "AuthFlow": "USER_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser"  
    },  
    "ClientId": "1example23456789"  
}
```

Amazon Cognito responds includes an `AvailableChallenges` parameter.

```
{  
    "AvailableChallenges": [  
        "EMAIL OTP",  
        "SMS OTP",  
        "PASSWORD"  
    ],  
    "Session": "[Session ID from the previous response]"  
}
```

This user is eligible for passwordless sign-in with email message OTP, SMS message OTP, and username-password. Your application can prompt the user for their selection, or make a selection based on internal logic. It then proceeds with a [RespondToAuthChallenge](#) or [AdminRespondToAuthChallenge](#) request that selects the challenge. Suppose the user wants to complete passwordless authentication with an email-message OTP.

```
{  
    "ChallengeName": "SELECT_CHALLENGE",  
    "ChallengeResponses": {
```

```
"USERNAME" : "testuser",
"ANSWER" : "EMAIL_OTP"
},
"ClientId": "lexample23456789",
"Session": "[Session ID from the previous response]"
}
```

Amazon Cognito responds with an EMAIL OTP challenge and sends a code to your user's verified email address. Your application then must respond again to this challenge.

```
{
  "ChallengeName": "EMAIL_OTP",
  "ChallengeResponses": {
    "USERNAME" : "testuser",
    "EMAIL_OTP_CODE" : "123456"
  },
  "ClientId": "lexample23456789",
  "Session": "[Session ID from the previous response]"
}
```

Passkey sign-in

Passkeys are secure and impose a relatively low effort level on users. Passkey sign-in makes use of *authenticators*, external devices that users can authenticate with. Regular passwords expose users to vulnerabilities like phishing, password guessing, and credential theft. With passkeys, your application can benefit from advanced security measures on mobile phones and other devices attached to or built in to information systems. A common passkey sign-in workflow starts with a call to your device that invokes your password or *credentials* manager, for example the iOS keychain or the Google Chrome password manager. The on-device credentials manager prompts them to select a passkey and authorize it with an existing credential or device-unlock mechanism. Modern phones have face scanners, fingerprint scanners, unlock patterns and other mechanisms, some that simultaneously satisfy the *something you know* and *something you have* principles of strong authentication. In the case of passkey authentication with biometrics, passkeys represent a *something you are*.

You might want to replace passwords with the thumbprint, face, or security-key authentication. This is *passkey* or *WebAuthn* authentication. It's common for application developers to permit users to enroll a biometric device after they first sign in with a password. With Amazon Cognito user pools, your application can configure this sign-in option for users.

What are passkeys?

Passkeys simplify the user experience by eliminating the need to remember complex passwords or enter OTPs. Passkeys are based on WebAuthn and CTAP2 standards drafted by the [World Wide Web Consortium](#) (W3C) and FIDO (Fast Identity Online) Alliance. Browsers and platforms implement these standards, provide APIs for web or mobile applications to start a passkey registration or authentication process, and also UI for user to select and interact with a passkey authenticator.

When a user registers an authenticator with a website or an app, the authenticator creates a public-private key pair. WebAuthn browsers and platforms submit the public key to the application back end of the website or app. The authenticator keeps the private key, key IDs, and metadata about the user and application. When the user wants to authenticate in the registered application with their registered authenticator, the application generates a random challenge. The response to this challenge is the digital signature of the challenge generated with the private key of the authenticator for that application and user, and relevant metadata. The browser or application platform receives the digital signature and passes it to the application back end. The application then validates the signature with the stored public key.

 **Note**

Your application doesn't receive any authentication secrets that users provide to their authenticator, nor does it receive information about the private key.

The following are some examples and capabilities of authenticators currently on the market. An authenticator might meet any or all of these categories.

- Some authenticators perform *user verification* with factors like a PIN, biometric input with a face or fingerprint, or a passcode before granting access, ensuring that only the legitimate user can authorize actions. Other authenticators don't have any user verification capabilities, and some can skip user verification when an application doesn't require it.
- Some authenticators, for example YubiKey hardware tokens, are portable. They communicate with devices through USB, Bluetooth or NFC connections. Some authenticators are local and bound to a platform, for example Windows Hello on a PC or Face ID on an iPhone. A device-bound authenticator can be carried by user if small enough, like a mobile device. Sometimes users can connect their hardware authenticator with many different platforms with wireless

communication. For example, users in desktop browsers can use their smart phone as a passkey authenticator when they scan a QR code.

- Some platform-bound passkeys sync to the cloud so that they can be used from multiple locations. For example, Face ID passkeys on iPhones sync passkey metadata with users' Apple accounts in their iCloud Keychain. These passkeys grant seamless authentication across Apple devices, instead of requiring that users register each device independently. Software-based authenticator apps like 1Password, Dashlane, and Bitwarden sync passkeys across all platforms where the user has installed the app.

In WebAuthn terminology, websites and apps are *relying parties*. Each passkey is associated with a specific relying party ID, a unified identifier that represents the websites or apps that accept passkey authentication.. Developers must carefully select their relying party ID to have the right scope of authentication. A typical relying party ID is the root domain name of a webserver. A passkey with this relying party ID specification can authenticate for that domain and subdomains. Browsers and platforms deny passkey authentication when the URL of the website a user want to access doesn't match the relying party ID. Similarly, for mobile apps, a passkey can only be used if the app path is present in the .well-known association files that the application makes available at the path indicated by the relying party ID.

Passkeys are *discoverable*. They can be automatically recognized and used by a browser or platform without requiring the user to input a username. When a user visits a website or app that supports passkey authentication, they can select from a list of passkeys that the browser or platform already knows, or they can scan a QR code.

How does Amazon Cognito implement passkey authentication?

Passkeys are an opt-in feature that's available in all [feature plans](#) other than **Lite**. It is only available in the [choice-based authentication flow](#). With [managed login](#), Amazon Cognito handles the logic of passkey authentication. You can also use the [Amazon Cognito user pools API in AWS SDKs](#) to do passkey authentication in your application back end.

Amazon Cognito recognizes passkeys created using either of two asymmetric cryptographic algorithms, ES256(-7) and RS256(-257). Most authenticators support both algorithms. By default, users can set up any type of authenticators, for example hardware tokens, mobile smart phones, and software authenticator apps. Amazon Cognito doesn't currently support [attestation enforcement](#).

In your user pool, you can configure user verification to be preferred or required. This setting defaults to preferred in API requests that don't provide a value, and preferred is selected by default in the Amazon Cognito console. When you set user verification to preferred, users can set up authenticators that don't have the user verification capability, and registration and authentication operations can succeed without user verification. To mandate user verification in passkey registration and authentication, change this setting to required.

The relying party (RP) ID that you set in your passkey configuration is an important decision. When you don't specify otherwise and your [domain branding version](#) is managed login, your user pool defaults to expecting the name of your [custom domain](#) as the RP ID. If you don't have a custom domain and don't specify otherwise, your user pool defaults to an RP ID of your [prefix domain](#). You can also configure your RP ID to be any domain name not in the public suffix list (PSL). Your RP ID entry applies to passkey registration and authentication in managed login and in SDK authentication. Passkey is only functional in mobile applications with Amazon Cognito can locate a .well-known association file with your RP ID as the domain. As a best practice, determine and set the value of your relying party ID before your website or app is publicly available. If you change your RP ID, your users must register again with the new RP ID.

Each user can register up to 20 passkeys. They can only register a passkey after they have signed in to your user pool at least once. Managed login removes significant effort from passkey registration. When you enable passkey authentication for a user pool and app client, your user pool with a managed login domain reminds end users to register a passkey after they sign up for a new user account. You can also invoke users' browsers at any time to direct them to a managed login page for passkey registration. Users must provide a username before Amazon Cognito can initiate passkey authentication. Managed login handles this automatically. The sign-in page prompts for a username, validates that the user has at least one passkey registered, and then prompts for passkey sign-in. Similarly, SDK-based applications must prompt for a username and provide it in the authentication request.

When you set up user pool authentication with passkeys and you have a custom domain and a prefix domain, the RP ID defaults to the fully-qualified domain name (FQDN) of your custom domain. To set a prefix domain as the RP ID in the Amazon Cognito console, delete your custom domain or enter the FQDN of the prefix domain as a **Third-party domain**.

Activate passkey sign-in

Console

To activate sign-in with passkeys, configure your user pool to permit primary sign-in with one or more passwordless types, then configure your app client to permit the USER_AUTH flow. In the Amazon Cognito console, navigate to the **Sign-in** menu under **Authentication** in your user pool configuration. Edit **Options for choice-based sign-in** and add **Passkey** to the list of **Available choices**.

Navigate to the **Authentication methods** menu and edit **Passkey**.

- **User verification** is the setting for whether your user pool requires passkey devices that perform additional checks that the current user is authorized for a passkey. To encourage users to configure a device with user verification, but not require it, select **Preferred**. To only support devices with user verification, select **Required**. For more information, see [User verification](#) at w3.org.
- The **Domain for relying party ID** is the identifier that your application will pass in users' passkey registration requests. It sets the target of the trust relationship with the issuer of users' passkeys. Your relying party ID can be: the domain of your user pool if **Cognito domain**

The Amazon Cognito [prefix domain](#) of your user pool.

Custom domain

The [custom domain](#) of your user pool.

Third-party domain

The domain for applications that don't use the user pools managed login pages. This setting is typically associated with user pools that don't have a [domain](#) and perform authentication with an AWS SDK and the user pools API in the backend.

Navigate to the **App clients** menu and choose an app client or create a new one. Select **Edit** and under **Authentication flows**, choose **Select an authentication type at sign-in: ALLOW_USER_AUTH**.

API/SDK

In the user pools API, configure **SignInPolicy** with the appropriate passkey options in a [CreateUserPool](#) or [UpdateUserPool](#) request. The **WEB_AUTHN** option for passkey authentication must be accompanied by at least one other option. Passkey registration requires an existing authentication session.

```
"SignInPolicy": {  
    "AllowedFirstAuthFactors": [  
        "PASSWORD",  
        "WEB_AUTHN"  
    ]  
}
```

Configure your user-verification preference and RP ID in the WebAuthnConfiguration parameter of a [SetUserPoolMfaConfig](#) request. The RelyingPartyId, the intended target of passkey authentication outcomes, can be your user pool prefix or custom domain, or a domain of your own choosing.

```
"WebAuthnConfiguration": {  
    "RelyingPartyId": "example.auth.us-east-1.amazoncognito.com",  
    "UserVerification": "preferred"  
}
```

Configure your app client ExplicitAuthFlows with the required option in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) request.

```
"ExplicitAuthFlows": [  
    "ALLOW_USER_AUTH"  
]
```

Register a passkey (managed login)

Managed login handles user registration of passkeys. When passkey authentication is active in your user pool, Amazon Cognito prompts users to set up a passkey when they register for a new user account.

Amazon Cognito doesn't prompt users to set up a passkey when they have already signed up and not set up a passkey, or if you created their account as an administrator. Users in this state must sign in with another factor like a password or passwordless OTP before they can register a passkey.

To register a passkey

1. Direct the user to your [sign-in page](#).

```
https://auth.example.com/oauth2/authorize/?  
client_id=1example23456789&response_type=code&scope=email+openid  
+phone&redirect_uri=https%3A%2F%2Fwww.example.com
```

2. Process the authentication result from the user. In this example, Amazon Cognito redirects them to www.example.com with an authorization code that your application exchanges for tokens.
3. Direct the user to your register-passkey page. The user will have a browser cookie that maintains their signed-in session. The passkey URL takes `client_id` and `redirect_uri` parameters. Amazon Cognito only permits authenticated users to access this page. Sign in your user with a password, email OTP, or SMS OTP and then invoke a URL that matches the following pattern.

You can also add other [Authorize endpoint](#) parameters to this request, like `response_type` and `scope`.

```
https://auth.example.com/passkeys/add?  
client_id=1example23456789&redirect_uri=https%3A%2F%2Fwww.example.com
```

Register a passkey (SDK)

You register passkey credentials with metadata in a [PublicKeyCreationOptions](#) object. You can generate this object with the credentials of a signed-in user and present them in an API request to their passkey issuer. The issuer will return a [RegistrationResponseJSON](#) object that confirms passkey registration.

To start the process of passkey registration, sign in a user with an existing sign-in option.

Authorize the [token-authorized StartWebAuthnRegistration](#) API request with the current user's access token. The following is the body of an example `GetWebAuthnRegistrationOptions` request.

```
{  
    "AccessToken": "eyJra456defEXAMPLE"  
}
```

The response from your user pool contains the PublicKeyCreationOptions object. Present this object in an API request to the user's issuer. It provides information like the public key and relying party ID. The issuer will respond with a RegistrationResponseJSON object.

Present the registration response in a [CompleteWebAuthnRegistration](#) API request, again authorized with the user's access token. When your user pool responds with an HTTP 200 response with an empty body, your user's passkey is registered.

Sign in with a passkey

Passwordless sign-in doesn't have an AuthFlow that you can specify in [InitiateAuth](#) and [AdminInitiateAuth](#). Instead, you must declare an AuthFlow of USER_AUTH and request a sign-in option or choose your passwordless option from the response from your user pool. To sign a user in to an application, configure the body of your InitiateAuth or AdminInitiateAuth request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

In this example, we know that the user wants to sign in with a passkey, and we add a PREFERRED_CHALLENGE parameter.

```
{  
    "AuthFlow": "USER_AUTH",  
    "AuthParameters": {  
        "USERNAME" : "testuser",  
        "PREFERRED_CHALLENGE" : "WEB_AUTHN"  
    },  
    "ClientId": "1example23456789"  
}
```

Amazon Cognito responds with a WEB_AUTHN challenge. Your application must respond to this challenge. Initiate a sign-in request with the user's passkey provider. It will return an [AuthenticationResponseJSON](#) object.

```
{  
    "ChallengeName": "WEB_AUTHN",  
    "ChallengeResponses": {  
        "USERNAME" : "testuser",  
        "CREDENTIAL" : "{AuthenticationResponseJSON}"  
    },  
    "ClientId": "1example23456789",  
    "Session": "[Session ID from the previous response]"
```

}

MFA after sign-in

You can set up users who complete sign-in with a username-password flow to be prompted for additional verification with a one-time password from an email message, SMS message, or code-generating application. MFA is distinct from passwordless sign-in, a first authentication factor that doesn't include MFA. MFA in user pools is a challenge-response model, where a user first demonstrates they know the password, then they demonstrate that they have access to their registered second-factor device.

Implementation resources

- [Adding MFA to a user pool](#)

Refresh tokens

When you want to permit users to select a *Remember me* checkbox, *refresh tokens* are the tool that your application has to persist a user's session. Applications can present refresh tokens to your user pool and exchange them for new ID and access tokens. With token refresh, you can ensure that a signed-in user is still active, get updated attribute information, and update access-control entitlements without user intervention.

Implementation resources

- [Ending user sessions with token revocation](#)

Custom authentication

You might want to configure a method of authentication for your users that isn't listed here. You can do that with *custom authentication* with Lambda triggers. In a sequence of Lambda functions, Amazon Cognito issues a challenge, asks a question that users must answer, checks the answer for accuracy, then determines if another challenge should be issued. The questions and answers can include security questions, requests to a CAPTCHA service, requests to an external MFA service API, or all of these in sequence.

Implementation resources

- [Custom authentication challenge Lambda triggers](#)

Custom authentication flow

Amazon Cognito user pools also make it possible to use custom authentication flows, which can help you create a challenge/response-based authentication model using AWS Lambda triggers.

The custom authentication flow makes possible customized challenge and response cycles to meet different requirements. The flow starts with a call to the `InitiateAuth` API operation that indicates the type of authentication to use and provides any initial authentication parameters. Amazon Cognito responds to the `InitiateAuth` call with one of the following types of information:

- 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 the user or app must first answer a challenge, but your custom code must determine this.)

If Amazon Cognito responds to the `InitiateAuth` call with a challenge, the app gathers more input and calls the `RespondToAuthChallenge` operation. This call provides the challenge responses and passes it back the session. Amazon Cognito responds to the `RespondToAuthChallenge` call similarly to the `InitiateAuth` call. If the user has signed in, Amazon Cognito provides tokens, or if the user isn't signed in, Amazon Cognito provides another challenge, or an error. If Amazon Cognito returns another challenge, the sequence repeats and the app calls `RespondToAuthChallenge` until the user successfully signs in or an error is returned. For more details about the `InitiateAuth` and `RespondToAuthChallenge` API operations, see the [API documentation](#).

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, three Lambda triggers control challenges and verification of the responses.

- The DefineAuthChallenge Lambda trigger uses a session array of previous challenges and responses as input. It then generates the next challenge name and Booleans that indicate whether the user is authenticated and can be granted tokens. 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. When DefineAuthChallenge returns CUSTOM_CHALLENGE as the next challenge, the authentication flow calls CreateAuthChallenge. The CreateAuthChallenge Lambda trigger passes the next type of challenge 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 must begin with SRP.

- To initiate SRP password verification in a custom flow, the app calls `InitiateAuth` with `CUSTOM_AUTH` as the `Authflow`. In the `AuthParameters` map, the request from your app includes `SRP_A`: (the SRP A value) and `CHALLENGE_NAME`: `SRP_A`.
- The `CUSTOM_AUTH` flow invokes the `DefineAuthChallenge` Lambda trigger with an initial session of `challengeName`: `SRP_A` and `challengeResult`: `true`. Your Lambda function responds with `challengeName`: `PASSWORD_VERIFIER`, `issueTokens`: `false`, and `failAuthentication`: `false`.
- The app next must call `RespondToAuthChallenge` with `challengeName`: `PASSWORD_VERIFIER` and the other parameters required for SRP in the `challengeResponses` map.
- If Amazon Cognito verifies the password, `RespondToAuthChallenge` invokes the `DefineAuthChallenge` Lambda trigger with a second session of `challengeName`: `PASSWORD_VERIFIER` and `challengeResult`: `true`. At that point, the `DefineAuthChallenge` Lambda trigger responds with `challengeName`: `CUSTOM_CHALLENGE` to start the custom challenge.

- If MFA is enabled for a user, after Amazon Cognito verifies the password, your user is then challenged to set up or sign in with MFA.

Note

The Amazon Cognito hosted sign-in webpage can't activate [Custom authentication challenge Lambda triggers](#).

For more information about the Lambda triggers, including sample code, see [Customizing user pool workflows with Lambda triggers](#).

Admin authentication flow

User migration authentication flow

A user migration Lambda trigger helps migrate users from a legacy user management system into your user pool. If you choose the USER_PASSWORD_AUTH authentication flow, users don't have to reset their passwords during user migration. This flow sends your users' passwords to the service over an encrypted SSL connection during authentication.

When you have migrated all your users, switch flows to the more secure SRP flow. The SRP flow doesn't send any passwords over the network.

To learn more about Lambda triggers, see [Customizing user pool workflows with Lambda triggers](#).

For more information about migrating users with a Lambda trigger, see [Importing users with a user migration Lambda trigger](#).

Authorization models for API and SDK authentication

When you start out with user pools authentication, you must decide on your application authorization model. Amazon Cognito authentication typically requires that you implement two API operations in order. The API operations that you use for authentication depend on the characteristics of your application. Public clients, where the application is distributed to users, use public authentication, where requests for sign-in don't require authorization. Server-side clients, where the application logic is hosted on a remote system, can protect authentication operations with IAM authorization for sign-in requests. The following API operation pairs and their corresponding SDK methods map to each of the available authorization models.

To compare API authentication and see a full list of API operations and their authorization models, see [Understanding API, OIDC, and managed login pages authentication](#).

Client-side (public) authentication

1. [InitiateAuth](#)
2. [RespondToAuthChallenge](#)

InitiateAuth and RespondToAuthChallenge are unauthenticated APIs for use with client-side public app clients. For more information, see [Client-side authentication options](#) and [Understanding API, OIDC, and managed login pages authentication](#).

Server-side authentication

1. [AdminInitiateAuth](#)
2. [AdminRespondToAuthChallenge](#)

AdminInitiateAuth and AdminRespondToAuthChallenge require IAM credentials and are suited for server-side confidential app clients. For more information, see [Server-side authentication options](#) and [Understanding API, OIDC, and managed login pages authentication](#).

A user authenticates by answering successive challenges until authentication either fails or Amazon Cognito issues tokens to the user. You can repeat these steps with Amazon Cognito, in a process that includes different challenges, to support any custom authentication flow.

Topics

- [Server-side authentication options](#)
- [Client-side authentication options](#)
- [Understanding API, OIDC, and managed login pages authentication](#)

Server-side authentication options

Web applications and other *server-side* applications implement authentication in a session on a remote server, usually in a browser that initiates a session to that server. Server-side applications typically have the following characteristics.

- They're built in an application installed on a server in languages like Java, Ruby, or Node.js.

- They connect to user pool [app clients](#) that might have a client secret, called *confidential clients*.
- They have access to AWS credentials.
- They invoke [managed login](#) for authentication, or use IAM-authorized operations in the user pools API with an AWS SDK.
- They serve internal customers and might serve public customers.

Server-side operations with the user pools API can use passwords, one-time passwords, or passkeys as the primary sign-in factor. For server-side apps, user pool authentication is similar to authentication for client-side apps, except for the following:

- The server-side app makes an [AdminInitiateAuth](#) API request. This operation requires AWS credentials with permissions that include cognito-*idp*:AdminInitiateAuth and cognito-*idp*:AdminRespondToAuthChallenge. The operation returns the required challenge or authentication outcome.
- When the application receives a challenge, it makes an [AdminRespondToAuthChallenge](#) API request. The AdminRespondToAuthChallenge API operation also requires AWS credentials.

For more information about signing Amazon Cognito API requests with AWS credentials, see [Signature Version 4 signing process](#) in the *AWS General Reference*.

In the AdminInitiateAuth response ChallengeParameters, the USER_ID_FOR_SRSP attribute, if present, contains the user's actual username, not an alias (such as email address or phone number). In your call to AdminRespondToAuthChallenge, in the ChallengeResponses, you must pass this username in the USERNAME parameter.

 **Note**

Because backend admin implementations use the admin authentication flow, the flow doesn't support remembered devices. When you have turned on device tracking, admin authentication succeeds, but any call to refresh the access token fails.

Client-side authentication options

Mobile apps and other *client-side* application types installed on users' devices typically have the following characteristics.

- They're built in languages like React native, Flutter, and Swift and deploy to user devices.
- They connect to user pool [app clients](#) that don't have a client secret, called *public clients*.
- They don't have access to AWS credentials that would authorize IAM-authorized API requests.
- They invoke [managed login](#) for authentication, or use public and token-authorized operations in the user pools API with an AWS SDK.
- They serve public customers and permit anyone to sign up and sign in.

Client-side operations with the user pools API can use passwords, one-time passwords, or passkeys as the primary sign-in factor. The following process works for user client-side apps that you create with [AWS Amplify](#) or the [AWS SDKs](#).

1. The user enters their username and password into the app.
2. The app calls the `InitiateAuth` operation with the user's username and Secure Remote Password (SRP) details.

This API operation returns the authentication parameters.

 **Note**

The app generates SRP details with the Amazon Cognito SRP features that are built in to AWS SDKs.

3. The app calls the `RespondToAuthChallenge` operation. If the call succeeds, Amazon Cognito returns the user's tokens, and the authentication flow is complete.

If Amazon Cognito requires another challenge, the call to `RespondToAuthChallenge` returns no tokens. Instead, the call 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).

Understanding API, OIDC, and managed login pages authentication

Amazon Cognito user pools are a combination of several authentication technologies. They are relying parties to external identity providers (IdPs). They are IdPs to applications that implement authentication with OpenID Connect (OIDC) SDKs. They provide authentication as issuers of JSON

web tokens (JWTs) similar to OIDC authentication, but in API methods that are part of AWS SDKs. They can also be secure points of entry to your applications.

When you want to sign up, sign in, and manage users in your user pool, you have two options.

1. Your *managed login pages* and the classic *hosted UI* include the [managed login user-interactive endpoints](#) and the [federation endpoints](#) that handle IdP and relying-party roles. They make up a package of public webpages that Amazon Cognito activates when you [choose a domain](#) for your user pool. For a quick start with the authentication and authorization features of Amazon Cognito user pools, including pages for sign-up, sign-in, password management, and multi-factor authentication (MFA), use the built-in user interface of managed login.

The other user pool endpoints facilitate authentication with third-party identity providers (IdPs). The services that they perform include the following.

- a. Service-provider callback endpoints for authenticated claims from your IdPs, like saml2/idpreponse and oauth2/idpreponse. When Amazon Cognito is an intermediate service provider (SP) between your app and your IdP, the callback endpoints represent the service.
 - b. Endpoints that provide information about your environment, like oauth2/userInfo and /.well-known/jwks.json. Your app uses these endpoints when it verifies tokens or retrieves user profile data with OIDC or OAuth 2.0 developer libraries.
2. The [Amazon Cognito user pools API](#) is a set of tools for your web or mobile app to authenticate users after it collects sign-in information in your own custom front end. User pools API authentication produces the following JSON web tokens.
 - a. An identity token with verifiable attribute claims from your user.
 - b. An access token that authorizes your user to create token-authorized API requests to an [AWS service endpoint](#).

 **Note**

By default, access tokens from user pools API authentication only contain the aws.cognito.signin.user.admin scope. To generate an access token with additional scopes, for example to authorize a request to a third-party API, request scopes during authentication through your user pool endpoints or add custom scopes in a [Pre token generation Lambda trigger](#). Access token customization adds costs to your AWS bill.

- c. A refresh token that authorizes requests for new ID and access tokens, and refreshes user identity and access-control properties.

You can link a federated user, who would normally sign in through the user pools endpoints, with a user whose profile is *local* to your user pool. A local user exists exclusively in your user pool directory without federation through an external IdP. If you link their federated identity to a local user in an [AdminLinkProviderForUser](#) API request, they can sign in with the user pools API. For more information, see [Linking federated users to an existing user profile](#).

The Amazon Cognito user pools API is dual-purpose.

1. It creates and configures your Amazon Cognito user pools resources. For example, you can create user pools, add AWS Lambda triggers, and configure the user pool domain that hosts your managed login pages.
2. It performs sign-up, sign-in and other user operations for local and linked users.

Example scenario with the Amazon Cognito user pools API

1. Your user selects a "Create an account" button that you created in your app. They enter an email address and password.
2. Your app sends a [SignUp](#) API request and creates a new user in your user pool.
3. Your app prompts your user for an email confirmation code. Your users enters the code they received in an email message.
4. Your app sends a [ConfirmSignUp](#) API request with the user's confirmation code.
5. Your app prompts your user for their username and password, and they enter their information.
6. Your app sends an [InitiateAuth](#) API request and stores an ID token, access token, and refresh token. Your app calls OIDC libraries to manage your user's tokens and maintain a persistent session for that user.

In the Amazon Cognito user pools API, you can't sign in users who federate through an IdP. You must authenticate these users through your user pool endpoints. For more information about the user pool endpoints that include managed login, see [User pool endpoints and managed login reference](#).

Your federated users can start in managed login and select their IdP, or you can skip managed login and send your users directly to your IdP to sign in. When your API request to the [Authorize endpoint](#) includes an IdP parameter, Amazon Cognito silently redirects your user to the IdP sign-in page.

Example scenario with managed login pages

1. Your user selects a "Create an account" button that you created in your app.
2. Managed login presents your user with a list of the social identity providers where you have registered developer credentials. Your user chooses Apple.
3. Your app initiates a request to the [Authorize endpoint](#) with provider name SignInWithApple.
4. Your user's browser opens the Apple authentication page. Your user signs in and chooses to authorize Amazon Cognito to read their profile information.
5. Amazon Cognito confirms the Apple access token and queries your user's Apple profile.
6. Your user presents an Amazon Cognito authorization code to your app.
7. The OIDC library in your application exchanges the authorization code with the [Token endpoint](#) and stores an ID token, access token, and refresh token issued by the user pool. Your app uses OIDC libraries to manage your user's tokens and maintain a persistent session for that user.

The user pools API and managed login pages support a variety of scenarios, described throughout this guide. The following sections examine how the user pools API further divides into classes that support your sign-up, sign-in, and resource-management requirements.

Amazon Cognito user pools authenticated and unauthenticated API operations

The Amazon Cognito user pools API, both a resource-management interface and a user-facing authentication and authorization interface, combines the authorization models that follow in its operations. Depending on the API operation, you might have to provide authorization with IAM credentials, an access token, a session token, a client secret, or a combination of these. For many user authentication and authorization operations, you have a choice of authenticated and unauthenticated versions of the request. Unauthenticated operations are best security practice for apps that you distribute to your users, like mobile apps; you don't need to include any secrets in your code.

You can only assign permissions in IAM policies for [IAM-authenticated management operations](#) and [IAM-authenticated user operations](#).

IAM-authenticated management operations

IAM-authenticated management operations modify and view your user pool and app client configuration, like you would do in the AWS Management Console.

For example, to modify your user pool in an [UpdateUserPool](#) API request, you must present AWS credentials and IAM permissions to update the resource.

To authorize these requests in the AWS Command Line Interface (AWS CLI) or an AWS SDK, configure your environment with environment variables or client configuration that adds IAM credentials to your request. For more information, see [Accessing AWS using your AWS credentials](#) in the *AWS General Reference*. You can also send requests directly to the [service endpoints](#) for the Amazon Cognito user pools API. You must authorize, or *sign*, these requests with AWS credentials that you embed in the header of your request. For more information, see [Signing AWS API requests](#).

IAM-authenticated management operations

AddCustomAttributes

CreateGroup

CreateIdentityProvider

CreateResourceServer

CreateUserImportJob

CreateUserPool

CreateUserPoolClient

CreateUserPoolDomain

DeleteGroup

DeleteIdentityProvider

DeleteResourceServer

DeleteUserPool

IAM-authenticated management operations

DeleteUserPoolClient

DeleteUserPoolDomain

DescribeIdentityProvider

DescribeResourceServer

DescribeRiskConfiguration

DescribeUserImportJob

DescribeUserPool

DescribeUserPoolClient

DescribeUserPoolDomain

GetCSVHeader

GetGroup

GetIdentityProviderByIdentifier

GetSigningCertificate

GetUICustomization

GetUserPoolMfaConfig

ListGroups

ListIdentityProviders

ListResourceServers

ListTagsForResource

ListUserImportJobs

ListUserPoolClients

IAM-authenticated management operations

ListUserPools

ListUsers

ListUsersInGroup

SetRiskConfiguration

SetUICustomization

SetUserPoolMfaConfig

StartUserImportJob

StopUserImportJob

TagResource

UntagResource

UpdateGroup

UpdateIdentityProvider

UpdateResourceServer

UpdateUserPool

UpdateUserPoolClient

UpdateUserPoolDomain

IAM-authenticated user operations

IAM-authenticated user operations sign up, sign in, manage credentials for, modify, and view your users.

For example, you can have a server-side application tier that backs a web front end. Your server-side app is an OAuth confidential client that you trust with privileged access to your Amazon Cognito resources. To register a user in the app, your server can include AWS credentials in an

[AdminCreateUser](#) API request. For more information about OAuth client types, see [Client Types](#) in *The OAuth 2.0 Authorization Framework*.

To authorize these requests in the AWS CLI or an AWS SDK, configure your server-side app environment with environment variables or client configuration that adds IAM credentials to your request. For more information, see [Accessing AWS using your AWS credentials](#) in the *AWS General Reference*. You can also send requests directly to the [service endpoints](#) for the Amazon Cognito user pools API. You must authorize, or *sign*, these requests with AWS credentials that you embed in the header of your request. For more information, see [Signing AWS API requests](#).

If your app client has a client secret, you must provide both your IAM credentials and, depending on the operation, either the SecretHash parameter or the SECRET_HASH value in AuthParameters. For more information, see [Computing secret hash values](#).

IAM-authenticated user operations

[AdminAddUserToGroup](#)

[AdminConfirmSignUp](#)

[AdminCreateUser](#)

[AdminDeleteUser](#)

[AdminDeleteUserAttributes](#)

[AdminDisableProviderForUser](#)

[AdminDisableUser](#)

[AdminEnableUser](#)

[AdminForgetDevice](#)

[AdminGetDevice](#)

[Admin GetUser](#)

[AdminInitiateAuth](#)

[AdminLinkProviderForUser](#)

IAM-authenticated user operations

AdminListDevices

AdminListGroupForUser

AdminListUserAuthEvents

AdminRemoveUserFromGroup

AdminResetUserPassword

AdminRespondToAuthChallenge

AdminSetUserMFAPreference

AdminSetUserPassword

AdminSetUserSettings

AdminUpdateAuthEventFeedback

AdminUpdateDeviceStatus

AdminUpdateUserAttributes

AdminUserGlobalSignOut

Unauthenticated user operations

Unauthenticated user operations sign up, sign in, and initiate password resets for your users. Use unauthenticated, or *public*, API operations when you want anyone on the internet to sign up and sign in to your app.

For example, to register a user in your app, you can distribute an OAuth public client that doesn't provide any privileged access to secrets. You can register this user with the unauthenticated API operation [SignUp](#).

To send these requests in a public client that you developed with an AWS SDK, you don't need to configure any credentials. You can also send requests directly to the [service endpoints](#) for the Amazon Cognito user pools API with no additional authorization.

If your app client has a client secret, you must provide, depending on the operation, either the `SecretHash` parameter or the `SECRET_HASH` value in `AuthParameters`. For more information, see [Computing secret hash values](#).

Unauthenticated user operations

`SignUp`

`ConfirmSignUp`

`ResendConfirmationCode`

`ForgotPassword`

`ConfirmForgotPassword`

`InitiateAuth`

Token-authorized user operations

Token-authorized user operations sign out, manage credentials for, modify, and view your users after they have signed in or begun the sign-in process. Use token-authorized API operations when you don't want to distribute secrets in your app, and you want to authorize requests with your user's own credentials. If your user has completed sign-in, you must authorize their token-authorized API request with an access token. If your user is in the middle of a sign-in process, you must authorize their token-authorized API request with a session token that Amazon Cognito returned in the response to the previous request.

For example, in a public client, you might want to update a user's profile in a way that restricts the write access to the user's own profile only. To make this update, your client can include the user's access token in a [UpdateUserAttributes](#) API request.

To send these requests in a public client that you developed with an AWS SDK, you don't need to configure any credentials. Include an `AccessToken` or `Session` parameter in your request. You can also send requests directly to the [service endpoints](#) for the Amazon Cognito user pools API. To authorize a request to a service endpoint, include the access or session token in the POST body of your request.

To sign an API request for a token-authorized operation, include the access token as an Authorization header in your request, in the format Bearer *<Base64-encoded access token>*.

Token-authorized user operations	AccessToken	Session
RespondToAuthChallenge	✓	
ChangePassword	✓	
GetUser	✓	
UpdateUserAttributes	✓	
DeleteUserAttributes	✓	
DeleteUser	✓	
ConfirmDevice	✓	
ForgetDevice	✓	
GetDevice	✓	
ListDevices	✓	
UpdateDeviceStatus	✓	
GetUserAttributeVerificationCode	✓	
VerifyUserAttribute	✓	

Token-authorized user operations	AccessToken	Session
SetUserSettings	✓	
SetUserMFAPreference	✓	
GlobalSignOut	✓	
AssociateSoftwareToken	✓	✓
UpdateAuthEventFeedback		✓
VerifySoftwareToken	✓	✓
RevokeToken ¹		

¹ RevokeToken takes a refresh token as a parameter. The refresh token serves as the authorizing token, and as the target resource.

Application resources for user pool authentication

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. The following are SDK examples for implementing authentication in your application.

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;  
}];
```

User pool sign-in with a third party identity provider

Your app users can either sign in directly through a user pool, or they can 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 IdPs. This way, your backend systems can standardize on one set of user pool tokens.

How federated sign-in works in Amazon Cognito user pools

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).



Amazon Cognito is a user directory and an OAuth 2.0 identity provider (IdP). When you sign in *local users* to the Amazon Cognito directory, your user pool is an IdP to your app. A local user exists exclusively in your user pool directory without federation through an external IdP.

When you connect Amazon Cognito to social, SAML, or OpenID Connect (OIDC) IdPs, your user pool acts as a bridge between multiple service providers and your app. To your IdP, Amazon Cognito is a service provider (SP). Your IdPs pass an OIDC ID token or a SAML assertion to Amazon Cognito. Amazon Cognito reads the claims about your user in the token or assertion and maps those claims to a new user profile in your user pool directory.

Amazon Cognito then creates a user profile for your federated user in its own directory. Amazon Cognito adds attributes to your user based on the claims from your IdP and, in the case of OIDC and social identity providers, an IdP-operated public `userinfo` endpoint. Your user's attributes change in your user pool when a mapped IdP attribute changes. You can also add more attributes independent of those from the IdP.

After Amazon Cognito creates a profile for your federated user, it changes its function and presents itself as the IdP to your app, which is now the SP. Amazon Cognito is a combination OIDC and OAuth 2.0 IdP. It generates access tokens, ID tokens, and refresh tokens. For more information about tokens, see [Understanding user pool JSON web tokens \(JWTs\)](#).

You must design an app that integrates with Amazon Cognito to authenticate and authorize your users, whether federated or local.

The responsibilities of an app as a service provider with Amazon Cognito

Verify and process the information in the tokens

In most scenarios, Amazon Cognito redirects your authenticated user to an app URL that it appends with an authorization code. Your app [exchanges the code](#) for access, ID, and refresh tokens. Then, it must [check the validity of the tokens](#) and serve information to your user based on the claims in the tokens.

Respond to authentication events with Amazon Cognito API requests

Your app must integrate with the [Amazon Cognito user pools API](#) and the [authentication API endpoints](#). The authentication API signs your user in and out, and manages tokens. The user pools API has a variety of operations that manage your user pool, your users, and the security of your authentication environment. Your app must know what to do next when it receives a response from Amazon Cognito.

Things to know about Amazon Cognito user pools third-party sign-in

- If you want your users to sign in with federated providers, you must choose a domain. This sets up the pages for [managed login](#). For more information, see [Using your own domain for managed login](#).
- You can't sign in federated users with API operations like [InitiateAuth](#) and [AdminInitiateAuth](#). Federated users can only sign in with the [Login endpoint](#) or the [Authorize endpoint](#).
- The [Authorize endpoint](#) is a *redirection* endpoint. If you provide an `idp_identifier` or `identity_provider` parameter in your request, it redirects silently to your IdP, bypassing managed login. Otherwise, it redirects to the managed login [Login endpoint](#).
- When managed login redirects a session to a federated IdP, Amazon Cognito includes the `user-agent` header `Amazon/Cognito` in the request.
- Amazon Cognito derives the `username` attribute for a federated user profile from a combination of a fixed identifier and the name of your IdP. To generate a user name that matches your custom requirements, create a mapping to the `preferred_username` attribute. For more information, see [Things to know about mappings](#).

Example: MyIDP_bob@example.com

- Amazon Cognito records information about your federated user's identity to an attribute, and a claim in the ID token, called **identities**. This claim contains your user's provider and their unique ID from the provider. You can't change the **identities** attribute in a user profile directly. For more information about how to link a federated user, see [Linking federated users to an existing user profile](#).
- When you update your IdP in an [UpdateIdentityProvider](#) API request, your changes can take up to a minute to appear in managed login.
- Amazon Cognito supports up to 20 HTTP redirects between itself and your IdP.
- When your user signs in with managed login, their browser stores an encrypted login-session cookie which records the client and provider they signed in with. If they attempt to sign in again with the same parameters, managed login reuses any *unexpired* existing session, and the user authenticates without providing credentials again. If your user signs in again with a different IdP, including a switch to or from local user pool sign-in, they must provide credentials and generate a new login session.

You can assign any of your user pool IdPs to any app client, and users can only sign in with an IdP that you assigned to their app client.

Topics

- [Configuring identity providers for your user pool](#)
- [Using social identity providers with a user pool](#)
- [Using SAML identity providers with a user pool](#)
- [Using OIDC identity providers with a user pool](#)
- [Mapping IdP attributes to profiles and tokens](#)
- [Linking federated users to an existing user profile](#)

Configuring identity providers for your user pool

With user pools, you can implement sign-in through a variety of external identity providers (IdPs). This section of the guide has instructions for setting up these identity providers with your user pool in the Amazon Cognito console. Alternatively, you can use the user pools API and an AWS SDK to programmatically add user pool identity providers. For more information, see [CreateIdentityProvider](#).

The supported identity provider options include social providers like Facebook, Google, and Amazon, as well as OpenID Connect (OIDC) and SAML 2.0 providers. Before you get started, set yourself up with administrative credentials for your IdP. For each type of provider, you'll need to register your application, obtain the necessary credentials, and then configure the provider details in your user pool. Your users can then sign up and sign in to your application with their existing accounts from the connected identity providers.

The **Social and external providers** menu under **Authentication** adds and updates user pool IdPs. For more information, see [User pool sign-in with a third party identity provider](#).

Topics

- [Set up user sign-in with a social IdP](#)
- [Set up user sign-in with an OIDC IdP](#)
- [Set up user sign-in with a SAML IdP](#)

Set up user sign-in with a social IdP

You can use federation to integrate Amazon Cognito user pools 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. After you have your developer account, 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.

- [Google identity platform](#)
- [Facebook for developers](#)
- [Login with Amazon](#)
- [Sign in with Apple](#)

To integrate user sign-in with a social IdP

1. Sign in to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.

4. Choose **Add an identity provider**, or choose the **Facebook**, **Google**, **Amazon**, or **Apple** identity provider you have configured, locate **Identity provider information**, and choose **Edit**. For more information about adding a social identity provider, see [Using social identity providers with a user pool](#).
5. Enter your social identity provider's information by completing one of the following steps, based on your choice of IdP:

Facebook, Google, and Login with Amazon

Enter the app ID and app secret that you received when you created your client app.

Sign In with Apple

Enter the service ID that you provided to Apple, and the team ID, key ID, and private key you received when you created your app client.

6. For **Authorized 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, that you want to access with your app. When entering scopes, use the following guidelines based on your choice of IdP:
 - **Facebook** — Separate scopes with commas. For example:
`public_profile, email`
 - **Google, Login with Amazon, and Sign In with Apple** — Separate scopes with spaces. For example:
 - **Google**: `profile email openid`
 - **Login with Amazon**: `profile postal_code`
 - **Sign In with Apple**: `name email`

 **Note**

For Sign In with Apple (console), use the check boxes to choose scopes.

7. Choose **Save changes**.
8. From the **App clients** menu, choose an app client from the list and then select **Edit**. Add the new social identity provider to the app client under **Identity providers**.
9. Choose **Save changes**.

For more information on social IdPs, see [Using social identity providers with a user pool](#).

Set up user sign-in with an OIDC IdP

You can integrate user sign-in with an OpenID Connect (OIDC) identity provider (IdP) such as Salesforce or Ping Identity.

To add an OIDC provider to a user pool

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools** from the navigation menu.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Social and external providers** menu and select **Add an identity provider**.
5. Choose an **OpenID Connect** identity provider.
6. Enter a unique name into **Provider name**.
7. Enter the client ID that you received from your provider into **Client ID**.
8. Enter the client secret that you received from your provider into **Client secret**.
9. Enter **Authorized scopes** for this provider. Scopes define which groups of user attributes (such as name and email) that your application will request from your provider. Scopes must be separated by spaces, following the [OAuth 2.0](#) specification.

Your user must consent to provide these attributes to your application.

10. Choose an **Attribute request method** to provide Amazon Cognito with the HTTP method (either GET or POST) that Amazon Cognito uses to fetch the details of the user from the **userInfo** endpoint operated by your provider.
11. Choose a **Setup method** to retrieve OpenID Connect endpoints either by **Auto fill through issuer URL** or **Manual input**. Use **Auto fill through issuer URL** when your provider has a public `.well-known/openid-configuration` endpoint where Amazon Cognito can retrieve the URLs of the authorization, token, userInfo, and jwks_uri endpoints.
12. Enter the issuer URL or authorization, token, userInfo, and jwks_uri endpoint URLs from your IdP.

Note

You can use only port numbers 443 and 80 with discovery, auto-filled, and manually entered URLs. User logins fail if your OIDC provider uses any nonstandard TCP ports.

The issuer URL must start with `https://`, and must not end with a / character. For example, Salesforce uses this URL:

`https://login.salesforce.com`

The openid-configuration document associated with your issuer URL must provide HTTPS URLs for the following values: `authorization_endpoint`, `token_endpoint`, `userinfo_endpoint`, and `jwks_uri`. Similarly, when you choose **Manual input**, you can only enter HTTPS URLs.

13. The OIDC claim **sub** is mapped to the user pool attribute **Username** by default. You can map other OIDC [claims](#) to user pool attributes. Enter the OIDC claim, and select the corresponding user pool attribute from the drop-down list. For example, the claim **email** is often mapped to the user pool attribute **Email**.
14. Map additional attributes from your identity provider to your user pool. For more information, see [Specifying Identity Provider attribute mappings for your user pool](#).
15. Choose **Create**.
16. From the **App clients** menu, select an app client from the list and select **Edit**. Add the new OIDC identity provider to the app client under **Identity providers**.
17. Choose **Save changes**.

For more information on OIDC IdPs, see [Using OIDC identity providers with a user pool](#).

Set up user sign-in with a SAML IdP

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 [Configuring your third-party SAML identity provider](#).

To configure a SAML 2.0 identity provider in your user pool

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Social and external provider** menu and select **Add an identity provider**.
5. Choose a **SAML** identity provider.

6. Enter **Identifiers** separated by commas. An identifier directs Amazon Cognito to check the user sign-in email address, and then direct the user to the provider that corresponds to their domain.
7. Choose **Add sign-out flow** if you want Amazon Cognito to send signed sign-out requests to your provider when a user logs out. Configure your SAML 2.0 identity provider to send sign-out responses to the <https://mydomain.us-east-1.amazoncognito.com/saml2/logout> endpoint that Amazon Cognito creates when you configure managed login. The saml2/logout endpoint uses POST binding.

 **Note**

If you select this option and your SAML identity provider expects a signed logout request, you also must 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.

8. Choose a **Metadata document source**. If your identity provider offers SAML metadata at a public URL, you can choose **Metadata document URL** and enter that public URL. Otherwise, choose **Upload metadata document** and select a metadata file you downloaded from your provider earlier.

 **Note**

If your provider has a public endpoint, we recommend that you enter a metadata document URL, rather than uploading a file. If you use the URL, Amazon Cognito refreshes metadata automatically. Typically, metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

9. **Map attributes between your SAML provider and your app** to map SAML provider attributes to the user profile in your user pool. Include your user pool required attributes in your attribute map.

For example, when you choose **User pool attribute email**, enter the SAML attribute name as it appears in the SAML assertion from your identity provider. Your identity provider might offer sample SAML assertions for reference. Some identity providers use simple names, such as `email`, while others use URL-formatted attribute names similar to:

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

10. Choose **Create**.

Note

If you see `InvalidParameterException` while creating a SAML IdP with an HTTPS metadata endpoint URL, make sure that the metadata endpoint has SSL correctly set up and that there is a valid SSL certificate associated with it. One example of such an exception would be "Error retrieving metadata from <`metadata endpoint`>".

To set up the SAML IdP to add a signing certificate

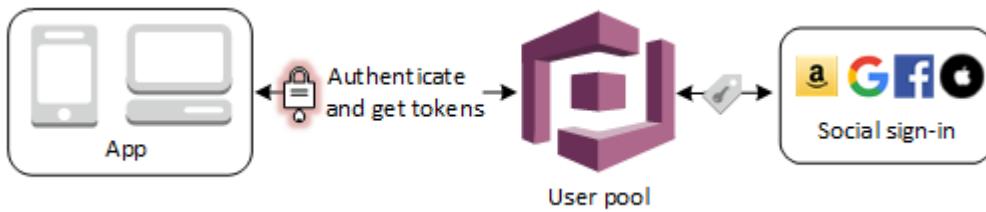
- To get the certificate containing the public key that the IdP uses to verify the signed logout request, do the following:
 1. Go to the **Social and external providers** menu of your user pool.
 2. Select your SAML provider,
 3. Choose **View signing certificate**.

For more information on SAML IdPs see [Using SAML identity providers with a user pool](#).

Using social identity providers with 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. This way, your backend systems can standardize on one set of user pool tokens. You must enable managed login to integrate with supported social identity providers. When Amazon Cognito builds your managed login pages, it creates OAuth 2.0 endpoints that Amazon Cognito and your OIDC and social IdPs use to exchange information. For more information, see the [Amazon Cognito user pools Auth API reference](#).

You can add a social IdP in the AWS Management Console, or you can use the AWS CLI or Amazon Cognito API.



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](#)
- [Step 1: Register with a social IdP](#)
- [Step 2: Add a social IdP to your user pool](#)
- [Step 3: Test your social IdP configuration](#)

Prerequisites

Before you begin, you need the following:

- A user pool with an app client and a user pool domain. For more information, see [Create a user pool](#).
- A social IdP.

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. Enter a name for your Facebook app, and then 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**, enter the path to the sign-in page for your app into **Site URL**.

```
https://mydomain.us-east-1.amazonaws.com/login?  
response_type=code&client_id=1example23456789&redirect_uri=https://www.example.com
```

10. Choose **Save changes**.
11. Enter the path to the root of your user pool domain into **App Domains**.

```
https://mydomain.us-east-1.amazonaws.com
```

12. Choose **Save changes**.
13. From the navigation bar choose **Add Product** and choose **Set up** for the **Facebook Login** product.
14. From the navigation bar choose **Facebook Login** and then **Settings**.

Enter the path to the /oauth2/idpresponse endpoint for your user pool domain into **Valid OAuth Redirect URIs**.

```
https://mydomain.us-east-1.amazonaws.com/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. Enter 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 icon and choose **Web Settings**, and then choose **Edit**.
9. Enter your user pool domain into **Allowed Origins**.

`https://mydomain.us-east-1.amazonaws.com`

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

`https://mydomain.us-east-1.amazonaws.com/oauth2/idpresponse`

11. Choose **Save**.

To register an app with Google

For more information about OAuth 2.0 in the Google Cloud platform, see [Learn about authentication & authorization](#) in the Google Workspace for Developers documentation.

1. Create a [developer account with Google](#).
2. Sign in to the [Google Cloud Platform console](#).
3. From the top navigation bar, choose **Select a project**. If you already have a project in the Google platform, this menu displays your default project instead.
4. Select **NEW PROJECT**.
5. Enter a name for your product and then choose **CREATE**.
6. On the left navigation bar, choose **APIs and Services**, then **Oauth consent screen**.
7. Enter App information, an **App domain**, **Authorized domains**, and **Developer contact information**. Your **Authorized domains** must include `amazoncognito.com` and the root of your custom domain, for example `example.com`. Choose **SAVE AND CONTINUE**.

8. 1. Under **Scopes**, choose **Add or remove scopes**, and choose, at minimum, the following OAuth scopes.
 1./auth/userinfo.email
 2./auth/userinfo.profile
 3. openid
9. Under **Test users**, choose **Add users**. Enter your email address and any other authorized test users, then choose **SAVE AND CONTINUE**.
10. Expand the left navigation bar again, and choose **APIs and Services**, then **Credentials**.
11. Choose **CREATE CREDENTIALS**, then **OAuth client ID**.
12. Choose an **Application type** and give your client a **Name**.
13. Under **Authorized JavaScript origins**, choose **ADD URI**. Enter your user pool domain.

`https://mydomain.us-east-1.amazonaws.com`

14. Under **Authorized redirect URIs**, choose **ADD URI**. Enter the path to the /oauth2/idpresponse endpoint of your user pool domain.

`https://mydomain.us-east-1.amazonaws.com/oauth2/idpresponse`

15. Choose **CREATE**.
16. Securely store the values the Google displays under **Your client ID** and **Your client secret**. Provide these values to Amazon Cognito when you add a Google IdP.

To register an app with Apple

For the most up-to-date information about setting up Sign in with Apple, see [Configuring Your Environment for Sign in with Apple](#) in the Apple Developer documentation.

1. Create a [developer account with Apple](#).
2. [Sign in](#) with your Apple credentials.
3. On the left navigation bar, choose **Certificates, Identifiers & 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 **Select a type** page, choose **App**, then choose **Continue**.
8. On the **Register an App ID** page, do the following:
 1. Under **Description**, enter a description.
 2. Under **App ID Prefix**, enter a **Bundle ID**. Make a note of the value under **App ID Prefix**. You will use this value after you choose Apple as your identity provider in [Step 2: Add a social IdP to your user pool](#).
 3. Under **Capabilities**, choose **Sign In with Apple**, and then choose **Edit**.
 4. On the **Sign in with Apple: App ID Configuration** page, choose to set up the app as either primary or grouped with other App IDs, and then choose **Save**.
 5. Choose **Continue**.
9. On the **Confirm your App ID** page, choose **Register**.
10. On the **Identifiers** page, choose the **+** 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:
 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](#).
 3. Choose **Continue**, then **Register**.
13. Choose the Services ID you just create from the Identifiers page.
 1. Select **Sign In with Apple**, and then choose **Configure**.
 2. On the **Web Authentication Configuration** page, select the app ID that you created earlier as the **Primary App ID**.
 3. Choose the **+** icon next to **Website URLs**.
 4. Under **Domains and subdomains**, enter your user pool domain without an `https://` prefix.

`mydomain.us-east-1.amazoncognito.com`
5. Under **Return URLs**, enter the path to the `/oauth2/idpresponse` endpoint of your user pool domain.

<https://mydomain.us-east-1.amazoncognito.com/oauth2/idpresponse>

6. Choose **Next**, and then **Done**. You don't need to verify the domain.
7. Choose **Continue**, and then choose **Save**.
14. On the left navigation bar, choose **Keys**.
15. On the **Keys** page, choose the + icon.
16. On the **Register a New Key** page, do the following:
 1. Under **Key Name**, enter a key name.
 2. Choose **Sign In with Apple**, and then choose **Configure**.
 3. On the **Configure Key** page and select the app ID that you created earlier as the **Primary App ID**. Choose **Save**.
 4. Choose **Continue**, and then choose **Register**.
17. On the **Download Your Key** page, choose **Download** to download the private key and note the **Key ID** shown, 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](#).

Step 2: Add a social IdP to your user pool

To configure a user pool social IdP with the AWS Management Console

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list or create a user pool.
4. Choose the **Social and external providers** menu and then select **Add an identity provider**.
5. Choose a social IdP: **Facebook**, **Google**, **Login with Amazon**, or **Sign in with Apple**.
6. Choose from the following steps, based on your choice of social IdP:
 - **Google** and **Login with Amazon** — Enter the **app client ID** and **app client secret** generated in the previous section.
 - **Facebook** — Enter the **app client ID** and **app client secret** generated in the previous section, and then choose an API version (for example, version 2.12). We recommend that you choose the latest possible version, as each Facebook API has a lifecycle and discontinuation date.

Facebook scopes and attributes can vary between API versions. We recommend that you test your social identity log in with Facebook to make sure that federation works as you intend.

- **Sign In with Apple** — Enter the **Services ID**, **Team ID**, **Key ID**, and **private key** generated in the previous section.
7. Enter the names of the **Authorized scopes** you want to use. 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 prompted to consent to providing these attributes to your app. For more information about social provider scopes, see the documentation from Google, Facebook, Login with Amazon, or Sign in with Apple.

With 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 Amazon 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. Map attributes from your IdP to your user pool. For more information, see [Specifying Identity Provider Attribute Mappings for Your User Pool](#).
9. Choose **Create**.

10. From the **App clients** menu, select an app client from the list and select **Edit**. Add the new social identity provider to the app client under **Identity providers**.
11. Choose **Save changes**.

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://mydomain.us-east-1.amazonaws.com/login?  
response_type=code&client_id=lexample23456789&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.

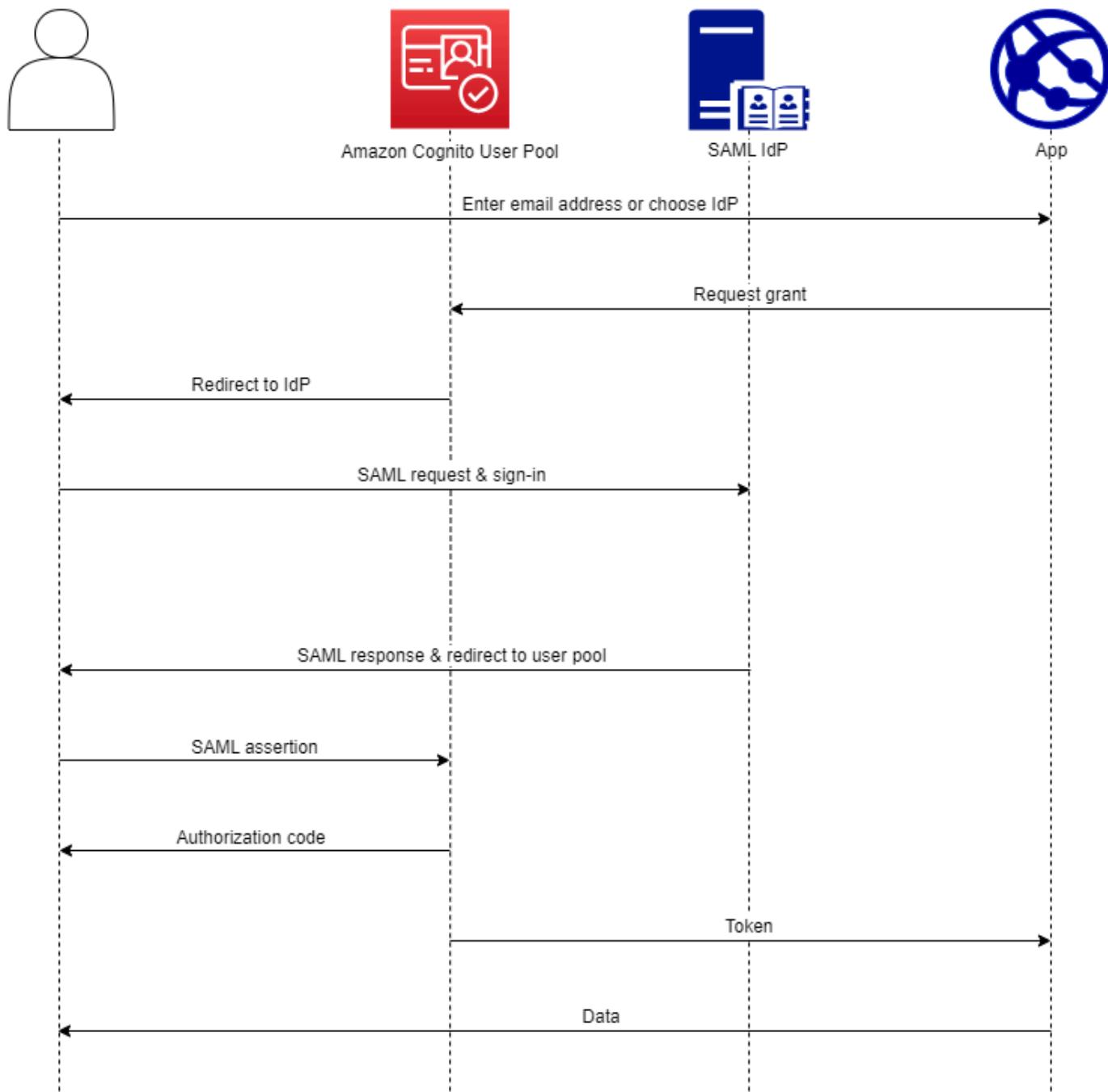
Note

Amazon Cognito cancels authentication requests that do not complete within 5 minutes, and redirects the user to managed login. The page displays a `Something went wrong` error message.

Using SAML identity providers with a user pool

You can choose to have your web and mobile app users sign in through a SAML identity provider (IdP) like [Microsoft Active Directory Federation Services \(ADFS\)](#), or [Shibboleth](#). You must choose a SAML IdP which supports the [SAML 2.0 standard](#).

With the managed login, Amazon Cognito authenticates local and third-party IdP users and issues JSON web tokens (JWTs). With the tokens that Amazon Cognito issues, you can consolidate multiple identity sources into a universal OpenID Connect (OIDC) standard across all of your apps. Amazon Cognito can process SAML assertions from your third-party providers into that SSO standard. You can create and manage a SAML IdP in the AWS Management Console, through the AWS CLI, or with the Amazon Cognito user pools API. To create your first SAML IdP in the AWS Management Console, see [Adding and managing SAML identity providers in a user pool](#).



Note

Federation with sign-in through a third-party IdP is a feature of Amazon Cognito user pools. Amazon Cognito identity pools, sometimes called Amazon Cognito federated identities, are an implementation of federation that you must set up separately in

each identity pool. A user pool can be a third-party IdP to an identity pool. For more information, see [Amazon Cognito identity pools](#).

Quick reference for IdP configuration

You must configure your SAML IdP to accept requests and send responses to your user pool. The documentation for your SAML IdP will contain information about how to add your user pool as a relying party or application for your SAML 2.0 IdP. The documentation that follows provides the values that you must provide for the SP entity ID and assertion consumer service (ACS) URL.

User pool SAML values quick reference

SP entity ID

`urn:amazon:cognito:sp:us-east-1_EXAMPLE`

ACS URL

`https://Your user pool domain/saml2/idpresponse`

You must configure your user pool to support your identity provider. The high-level steps to add an external SAML IdP are as follows.

1. Download SAML metadata from your IdP, or retrieve the URL to your metadata endpoint. See [Configuring your third-party SAML identity provider](#).
2. Add a new IdP to your user pool. Upload the SAML metadata or provide the metadata URL. See [Adding and managing SAML identity providers in a user pool](#).
3. Assign the IdP to your app clients. See [Application-specific settings with app clients](#)

Topics

- [Things to know about SAML IdPs in Amazon Cognito user pools](#)
- [Case sensitivity of SAML user names](#)
- [Configuring your third-party SAML identity provider](#)
- [Adding and managing SAML identity providers in a user pool](#)

- [SAML session initiation in Amazon Cognito user pools](#)
- [Signing out SAML users with single sign-out](#)
- [SAML signing and encryption](#)
- [SAML identity provider names and identifiers](#)

Things to know about SAML IdPs in Amazon Cognito user pools

Implementation of a SAML 2.0 IdP comes with some requirements and restrictions. Refer to this section when you're implementing your IdP. You'll also find information that's useful for troubleshooting errors during SAML federation with a user pool.

Amazon Cognito processes SAML assertions for you

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 IdP through a user agent. Your user pool acts as a service provider (SP) on behalf of your application. Amazon Cognito supports SP-initiated and IdP-initiated single sign-on (SSO) as described in sections 5.1.2 and 5.1.4 of the [SAML V2.0 Technical Overview](#).

Provide a valid IdP signing certificate

The signing certificate in your SAML provider metadata must not be expired when you configure the SAML IdP in your user pool.

User pools support multiple signing certificates

When your SAML IdP includes more than one signing certificate in SAML metadata, at sign-in your user pool determines that the SAML assertion is valid if it matches any certificate in the SAML metadata. Each signing certificate must be no longer than 4,096 characters in length.

Maintain the relay state parameter

Amazon Cognito and your SAML IdP maintain session information with a `relayState` parameter.

1. Amazon Cognito supports `relayState` values greater than 80 bytes. While 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` values greater than 80 bytes will break many standard SAML provider integrations.

2. The `relayState` token is an opaque reference to state information maintained by Amazon Cognito. Amazon Cognito doesn't guarantee the contents of the `relayState` parameter. Don't parse its contents such that your app depends on the result. For more information, see the [SAML 2.0 specification](#).

Identify the ACS endpoint

Your SAML identity provider requires that you set an assertion consumer endpoint. Your IdP redirects your users to this endpoint with their SAML assertion. Configure the following endpoint in your user pool domain for SAML 2.0 POST binding in your SAML identity provider.

`https://Your user pool domain/saml2/idpresponse`

With an Amazon Cognito domain:

`https://mydomain.us-east-1.amazoncognito.com/saml2/idpresponse`

With a custom domain:

`https://auth.example.com/saml2/idpresponse`

See [Configuring a user pool domain](#) for more information about user pool domains.

No replayed assertions

You can't repeat, or *replay*, a SAML assertion to your Amazon Cognito `saml2/idpresponse` endpoint. A replayed SAML assertion has an assertion ID that duplicates the ID of an earlier IdP response.

User pool ID is SP entity ID

You must provide your IdP with your user pool ID in the service provider (SP) `urn`, also called the *audience URI* or *SP entity ID*. The audience URI for your user pool has the following format.

`urn:amazon:cognito:sp:us-east-1_EXAMPLE`

You can find your user pool ID under **User pool overview** in the [Amazon Cognito console](#).

Map all required attributes

Configure your SAML IdP to provide values for any attributes that you set as required in your user pool. For example, `email` is a common required attribute for user pools. Before your users can sign in, your SAML IdP assertions must include a claim that you map to the **User pool attribute** `email`. For more information about attribute mapping, see [Mapping IdP attributes to profiles and tokens](#).

Assertion format has specific requirements

Your SAML IdP must include the following claims in the SAML assertion.

- A NameID claim. Amazon Cognito associates a SAML assertion with the destination user by NameID. If NameID changes, Amazon Cognito considers the assertion to be for a new user. The attribute that you set to NameID in your IdP configuration must have a persistent value. To assign SAML users to a consistent user profile in your user pool, assign your NameID claim from an attribute with a value that doesn't change.

```
<saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:persistent">
  carlos
</saml2:NameID>
```

A Format in your IdP NameID claim of `urn:oasis:names:tc:SAML:1.1:nameid-format:persistent` indicates that your IdP is passing an unchanging value. Amazon Cognito doesn't require this format declaration, and assigns a format of `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified` if your IdP doesn't specify a format of the NameID claim. This behavior complies with section 2.2.2 *Complex Type NameIDType*, of [the SAML 2.0 specification](#).

- An AudienceRestriction claim with an Audience value that sets your user pool SP entity ID as the target of the response.

```
<saml:AudienceRestriction>
  <saml:Audience> urn:amazon:cognito:sp:us-east-1_EXAMPLE
  </saml:Audience>
</saml:AudienceRestriction>
```

- For SP-initiated single sign-on, a Response element with an InResponseTo value of the original SAML request ID.

```
<saml2p:Response Destination="https://mydomain.us-east-1.amazonaws.com/
  saml2/idpreponse" ID="id123" InResponseTo="_dd0a3436-bc64-4679-
  a0c2-cb4454f04184" IssueInstant="Date-time stamp" Version="2.0"
  xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:xs="http://
  www.w3.org/2001/XMLSchema">
```

Note

IdP-initiated SAML assertions must *not* contain an InResponseTo value.

- A `SubjectConfirmationData` element with a `Recipient` value of your user pool saml2/idpreponse endpoint and, for SP-initiated SAML, an `InResponseTo` value that matches the original SAML request ID.

```
<saml2:SubjectConfirmationData InResponseTo="_dd0a3436-bc64-4679-a0c2-cb4454f04184" NotOnOrAfter="Date-time stamp" Recipient="https://mydomain.us-east-1.amazonaws.com/saml2/idpreponse"/>
```

SP-initiated sign-in requests

When the [Authorize endpoint](#) redirects your user to your IdP sign-in page, Amazon Cognito includes a *SAML request* in a URL parameter of the HTTP GET request. A SAML request contains information about your user pool, including your ACS endpoint. You can optionally apply a cryptographic signature to these requests.

Sign requests and encrypt responses

Every user pool with a SAML provider generates an asymmetric key pair and *signing certificate* for a digital signature that Amazon Cognito assigns to SAML requests. Every external SAML IdP that you configure to support encrypted SAML response causes Amazon Cognito to generate a new key pair and *encryption certificate* for that provider. To view and download the certificates with the public key, choose your IdP in the **Social and external providers** menu in the Amazon Cognito console.

To establish trust with SAML requests from your user pool, provide your IdP with a copy of your user pool SAML 2.0 signing certificate. Your IdP might ignore SAML requests that your user pool signed if you don't configure the IdP to accept signed requests.

1. Amazon Cognito applies a digital signature to SAML requests that your user passes to your IdP. Your user pool signs all single logout (SLO) requests, and you can configure your user pool to sign single sign-on (SSO) requests for any SAML external IdP. When you provide a copy of the certificate, your IdP can verify the integrity of your users' SAML requests.
2. Your SAML IdP can encrypt SAML responses with the encryption certificate. When you configure an IdP with SAML encryption, your IdP must only send encrypted responses.

Encode non-alphanumeric characters

Amazon Cognito doesn't accept 4-byte UTF-8 characters like # or # that your IdP passes as an attribute value. You can encode the character to Base64, pass it as text, and then decode it in your app.

In the following example, the attribute claim will not be accepted:

```
<saml2:Attribute Name="Name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xsd:string">#</saml2:AttributeValue>
</saml2:Attribute>
```

In contrast to the preceding example, the following attribute claim will be accepted:

```
<saml2:Attribute Name="Name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xsd:string">8J+YkA==</saml2:AttributeValue>
</saml2:Attribute>
```

Metadata endpoint must have valid transport-layer security

If you see `InvalidArgumentException` while creating a SAML IdP 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. For more information about validating certificates, see [What Is An SSL/TLS Certificate?](#).

App clients with IdP-initiated SAML can only sign in with SAML

When you activate support for a SAML 2.0 IdP that supports IdP-initiated sign in an app client, you can only add other SAML 2.0 IdPs to that app client. You're prevented from adding the user directory in the user pool *and* all non-SAML external identity providers to an app client configured in this way.

Logout responses must use POST binding

The `/saml2/logout` endpoint accepts `LogoutResponse` as HTTP POST requests. User pools don't accept logout responses with HTTP GET binding.

Case sensitivity of SAML user names

When a federated user attempts to sign in, the SAML identity provider (IdP) passes a unique `NameId` to Amazon Cognito in the user's SAML assertion. Amazon Cognito identifies a SAML-federated user by their `NameId` claim. Regardless of the case sensitivity settings of your user pool, Amazon Cognito recognizes a returning federated user from a SAML IdP when they pass their

unique and case-sensitive NameId claim. If you map an attribute like email to NameId, and your user changes their email address, they can't sign in to your app.

Map NameId in your SAML assertions from an IdP attribute that has values that don't change.

For example, Carlos has a user profile in your case-insensitive user pool from an Active Directory Federation Services (ADFS) SAML assertion that passed a NameId value of Carlos@example.com. The next time Carlos attempts to sign in, your ADFS IdP passes a NameId value of carlos@example.com. Because NameId must be an exact case match, the sign-in doesn't succeed.

If your users can't log in after their NameID changes, delete their user profiles from your user pool. Amazon Cognito will create new user profiles the next time they sign in.

Topics

- [Configuring your third-party SAML identity provider](#)
- [Adding and managing SAML identity providers in a user pool](#)
- [SAML session initiation in Amazon Cognito user pools](#)
- [Signing out SAML users with single sign-out](#)
- [SAML signing and encryption](#)
- [SAML identity provider names and identifiers](#)

Configuring your third-party SAML identity provider

When you want to add a SAML identity provider (IdP) to your user pool, you must make some configuration updates in the management interface of your IdP. This section describes how to format the values that you must provide to your IdP. You can also learn about how to retrieve the static or active-URL metadata document that identifies the IdP and its SAML claims to your user pool.

To configure third-party SAML 2.0 identity provider (IdP) solutions to work with federation for Amazon Cognito user pools, you must configure your SAML IdP to redirect to the following Assertion Consumer Service (ACS) URL: <https://mydomain.us-east-1.amazonaws.com/> saml2/idpreponse. If your user pool has an Amazon Cognito domain, you can find your user pool domain path in the **Domain** menu of your user pool in the [Amazon Cognito console](#).

Some SAML IdPs require that you provide the `urn`, also called the audience URI or SP entity ID, in the form `urn:amazon:cognito:sp:us-east-1_EXAMPLE`. You can find your user pool ID under **User pool overview** in the Amazon Cognito console.

You must also configure your SAML IdP to provide values for any attributes that you designated as *required attributes* in your user pool. Typically, `email` is a required attribute for user pools, in which case the SAML IdP must provide some form of an `email` claim in their SAML assertion, and you must map the claim to the attribute for that provider.

The following configuration information for third-party SAML 2.0 IdP solutions is a good place to start setting up federation with Amazon Cognito user pools. For the most current information, consult your provider's documentation directly.

To sign SAML requests, you must configure your IdP to trust requests signed by your user pool signing certificate. To accept encrypted SAML responses, you must configure your IdP to encrypt *all* SAML responses to your user pool. Your provider will have documentation about configuring these features. For an example from Microsoft, see [Configure Microsoft Entra SAML token encryption](#).

 **Note**

Amazon Cognito only requires your identity provider metadata document. Your provider might offer configuration information for AWS account federation with SAML 2.0; this information isn't relevant to Amazon Cognito integration.

Solution	More information
Microsoft Active Directory Federation Services (AD FS)	Federation Metadata Explorer
Okta	How to Download the IdP Metadata and SAML Signing Certificates for a SAML App Integration
Auth0	Configure Auth0 as SAML Identity Provider
Ping Identity (PingFederate)	Exporting SAML metadata from PingFederate
JumpCloud	SAML Configuration Notes

Solution	More information
SecureAuth	SAML application integration

Adding and managing SAML identity providers in a user pool

After you configure your identity provider to work with Amazon Cognito, you can add it to your user pools and app clients. The following procedures demonstrate how to create, modify, and delete SAML providers in an Amazon Cognito user pool.

AWS Management Console

You can use the AWS Management Console to create and delete SAML identity providers (IdPs).

Before you create a SAML IdP, you must have the SAML metadata document that you get from the third-party IdP. For instructions on how to get or generate the required SAML metadata document, see [Configuring your third-party SAML identity provider](#).

To configure a SAML 2.0 IdP in your user pool

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Social and external providers** menu and then select **Add an identity provider**.
5. Choose a **SAML** IdP.
6. Enter a **Provider name**. You can pass this friendly name in an `identity_provider` request parameter to the [Authorize endpoint](#).
7. Enter **Identifiers** separated by commas. An identifier tells Amazon Cognito it should check the email address a user enters when they sign in, and then direct them to the provider that corresponds to their domain.
8. Choose **Add sign-out flow** if you want Amazon Cognito to send signed sign-out requests to your provider when a user logs out. You must configure your SAML 2.0 IdP to send sign-out responses to the `https://mydomain.us-east-1.amazonaws.com/saml2/logout` endpoint that is created when you configure managed login. The `saml2/logout` endpoint uses POST binding.

Note

If this option is selected and your SAML IdP expects a signed logout request, you must also provide your SAML IdP with the signing certificate from your user pool. The SAML IdP will process the signed logout request and sign out your user from the Amazon Cognito session.

9. Choose your **IdP-initiated SAML sign-in** configuration. As a security best practice, choose **Accept SP-initiated SAML assertions only**. If you have prepared your environment to securely accept unsolicited SAML sign-in sessions, choose **Accept SP-initiated and IdP-initiated SAML assertions**. For more information, see [SAML session initiation in Amazon Cognito user pools](#).
10. Choose a **Metadata document source**. If your IdP offers SAML metadata at a public URL, you can choose **Metadata document URL** and enter that public URL. Otherwise, choose **Upload metadata document** and select a metadata file you downloaded from your provider earlier.

Note

We recommend that you enter a metadata document URL if your provider has a public endpoint instead of uploading a file. Amazon Cognito automatically refreshes metadata from the metadata URL. Typically, metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

11. **Map attributes between your SAML provider and your user pool** to map SAML provider attributes to the user profile in your user pool. Include your user pool required attributes in your attribute map.

For example, when you choose **User pool attribute** `email`, enter the SAML attribute name as it appears in the SAML assertion from your IdP. If your IdP offers sample SAML assertions, you can use these sample assertions to help you to find the name. Some IdPs use simple names, such as `email`, while others use names like the following.

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

12. Choose **Create**.

API/CLI

Use the following commands to create and manage a SAML identity provider (IdP).

To create an IdP and upload a metadata document

- AWS CLI: `aws cognito-identity-provider create-identity-provider`

Example with metadata file: `aws cognito-identity-provider create-identity-provider --user-pool-id us-east-1_EXAMPLE --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:

```
"ProviderDetails": {  
    "MetadataFile": "<SAML metadata XML>",  
    "IDPSignout" : "true",  
    "RequestSigningAlgorithm" : "rsa-sha256",  
    "EncryptedResponses" : "true",  
    "IDPInit" : "true"  
}
```

Note

If the `<SAML metadata XML>` contains any instances of the character ", you must add \ as an escape character: \".

Example with metadata URL: `aws cognito-identity-provider create-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1 --provider-type SAML --provider-details MetadataURL=https://myidp.example.com/sso/saml/metadata --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

- AWS API: [CreateIdentityProvider](#)

To upload a new metadata document for an IdP

- AWS CLI: aws cognito-idp update-identity-provider

Example with metadata file: aws cognito-idp update-identity-provider --user-pool-id *us-east-1_EXAMPLE* --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:

```
"ProviderDetails": {  
    "MetadataFile": "<SAML metadata XML>",  
    "IDPSignout" : "true",  
    "RequestSigningAlgorithm" : "rsa-sha256",  
    "EncryptedResponses" : "true",  
    "IDPInit" : "true"  
}
```

Note

If the *<SAML metadata XML>* contains any instances of the character ", you must add \ as an escape character: \".

Example with metadata URL: aws cognito-idp update-identity-provider --user-pool-id *us-east-1_EXAMPLE* --provider-name=*SAML_provider_1* --provider-details MetadataURL=*https://myidp.example.com/sso/saml/metadata* --attribute-mapping *email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress*

- AWS API: [UpdateIdentityProvider](#)

To get information about a specific IdP

- AWS CLI: aws cognito-idp describe-identity-provider

```
aws cognito-idp describe-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1
```

- AWS API: [DescribeldentityProvider](#)

To list information about all IdPs

- AWS CLI: aws cognito-idp list-identity-providers

Example: aws cognito-idp list-identity-providers --user-pool-id *us-east-1_EXAMPLE* --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 *us-east-1_EXAMPLE* --provider-name=*SAML_provider_1*

- AWS API: [DeleteIdentityProvider](#)

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:*us-east-1_EXAMPLE*. Amazon Cognito requires an audience restriction value that matches this URN in the SAML response. Configure your IdP to use the following POST binding endpoint for the IdP-to-SP response message.

`https://mydomain.us-east-1.amazonaws.com/saml2/idpreponse`

- Your SAML IdP must populate 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. Your IdP must pass each user's SAML name ID in a consistent, case-sensitive format. Any variation in the value of a user's name ID creates a new user profile.

To provide a signing certificate to your SAML 2.0 IDP

- To download a copy of the the public key from Amazon Cognito that your IdP can use to validate SAML logout requests, choose the **Social and external providers** menu of your user pool, select your IdP, and under **View signing certificate**, select **Download as .crt**.

You can delete any SAML provider you have set up in your user pool with the Amazon Cognito console.

To delete a SAML provider

1. Sign in to the [Amazon Cognito console](#).
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Select the radio button next to the SAML IdPs you wish to delete.
5. When you are prompted to **Delete identity provider**, enter the name of the SAML provider to confirm deletion, and then choose **Delete**.

SAML session initiation in Amazon Cognito user pools

Amazon Cognito supports service provider-initiated (SP-initiated) single sign-on (SSO) and IdP-initiated SSO. As a best security practice, implement SP-initiated SSO in your user pool. Section 5.1.2 of the [SAML V2.0 Technical Overview](#) describes SP-initiated SSO. Amazon Cognito is the identity provider (IdP) to your app. The app is the service provider (SP) that retrieves tokens for authenticated users. However, when you use a third-party IdP to authenticate users, Amazon Cognito is the SP. When your SAML 2.0 users authenticate with an SP-initiated flow, they must always first make a request to Amazon Cognito and redirect to the IdP for authentication.

For some enterprise use cases, access to internal applications starts at a bookmark on a dashboard hosted by the enterprise IdP. When a user selects a bookmark, the IdP generates a SAML response and sends it to the SP to authenticate the user with the application.

You can configure a SAML IdP in your user pool to support IdP-initiated SSO. When you support IdP-initiated authentication, Amazon Cognito can't verify that it has solicited the SAML response that it receives because Amazon Cognito doesn't initiate authentication with a SAML request. In SP-initiated SSO, Amazon Cognito sets state parameters that validate a SAML response against the original request. With SP-initiated sign-in you can also guard against cross-site request forgery (CSRF).

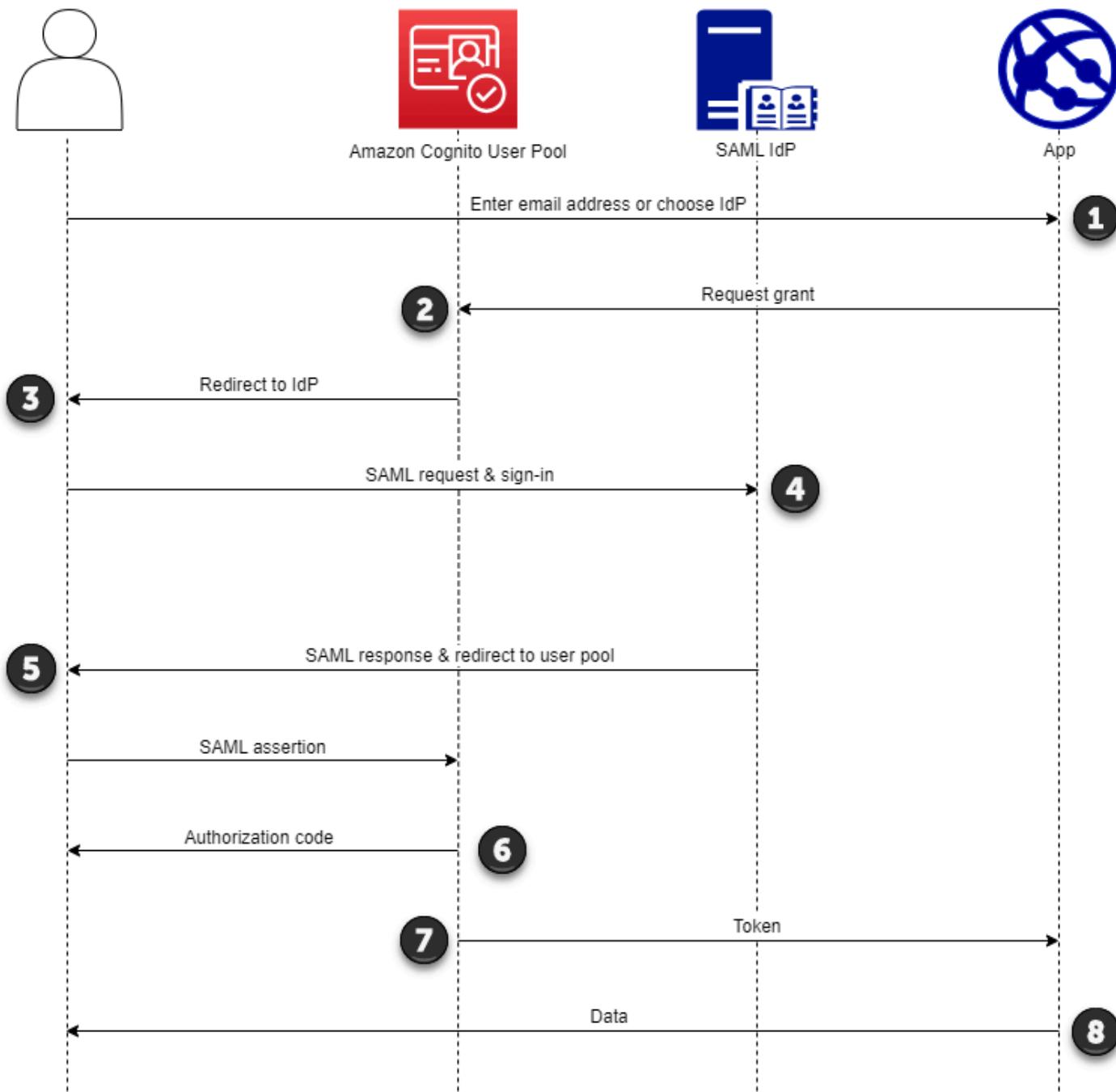
Topics

- [Using SP-initiated SAML sign-in](#)
- [Using IdP-initiated SAML sign-in](#)

Using SP-initiated SAML sign-in

As a best practice, implement service-provider-initiated (SP-initiated) sign-in to your user pool. Amazon Cognito initiates your user's session and redirects them to your IdP. With this method, you have the greatest control over who presents sign-in requests. You can also permit IdP-initiated sign-in under certain conditions.

The following process shows how users complete SP-initiated sign in to your user pool through a SAML provider.



1. Your user enters their email address at a sign-in page. To determine your user's redirect to their IdP, you can collect their email address in a custom-built application or invoke managed login in web view. You can configure your managed login pages to display a list of IdPs or to only prompt for an email address.
2. Your app invokes your user pool redirect endpoint and requests a session with the client ID that corresponds to the app and the IdP ID that corresponds to the user.

3. Amazon Cognito redirects your user to the IdP with a SAML request, [optionally signed](#), in an AuthnRequest element.
4. The IdP authenticates the user interactively, or with a remembered session in a browser cookie.
5. The IdP redirects your user to your user pool SAML response endpoint with the [optionally-encrypted](#) SAML assertion in their POST payload.

 **Note**

Amazon Cognito cancels sessions that don't receive a response within 5 minutes, and redirects the user to managed login. When your user experiences this outcome, they receive a Something went wrong error message.

6. After it verifies the SAML assertion and [maps user attributes](#) from the claims in the response, Amazon Cognito internally creates or updates the user's profile in the user pool. Typically, your user pool returns an authorization code to your user's browser session.
7. Your user presents their authorization code to your app, which exchanges the code for JSON web tokens (JWTs).
8. Your app accepts and processes your user's ID token as authentication, generates authorized requests to resources with their access token, and stores their refresh token.

When a user authenticates and receives an authorization code grant, the user pool returns ID, access, and refresh tokens. The ID token is a authentication object for OIDC-based identity management. The access token is an authorization object with [OAuth 2.0](#) scopes. The refresh token is an object that generates new ID and access tokens when your user's current tokens have expired. You can configure the duration of users' tokens in your user pool app client.

You can also choose the duration of refresh tokens. After a user's refresh token expires, they must sign in again. If they authenticated through a SAML IdP, your users' session duration is set by the expiration of their tokens, not the expiration of their session with their IdP. Your app must store each user's refresh token and renew their session when it expires. Managed login maintains user sessions in a browser cookie that's valid for 1 hour.

Using IdP-initiated SAML sign-in

When you configure your identity provider for IdP-initiated SAML 2.0 sign-in, you can present SAML assertions to the saml2/idpreponse endpoint in your user pool domain without the need to initiate the session at the [Authorize endpoint](#). A user pool with this configuration accepts IdP-

initiated SAML assertions from a user pool external identity provider that the requested app client supports. The following steps describe the overall process to configure and sign in with an IdP-initiated SAML 2.0 provider.

1. Create or designate a user pool and app client.
2. Create a SAML 2.0 IdP in your user pool.
3. Configure your IdP to support IdP initiation. IdP-initiated SAML introduces security considerations that other SSO providers aren't subject to. Because of this, you can't add non-SAML IdPs, including the user pool itself, to any app client that uses a SAML provider with IdP-initiated sign-in.
4. Associate your IdP-initiated SAML provider with an app client in your user pool.
5. Direct your user to the sign-in page for your SAML IdP and retrieve a SAML assertion.
6. Direct your user to your user pool `saml2/idpreponse` endpoint with their SAML assertion.
7. Receive JSON web tokens (JWTs).

To accept unsolicited SAML assertions in your user pool, you must consider its effect on your app security. Request spoofing and CSRF attempts are likely when you accept IdP-initiated requests. Although your user pool can't verify an IdP-initiated sign-in session, Amazon Cognito validates your request parameters and SAML assertions.

Additionally, your SAML assertion must not contain an `InResponseTo` claim and must have been issued within the previous 6 minutes.

You must submit requests with IdP-initiated SAML to your `/saml2/idpreponse`. For SP-initiated and managed login authorization requests, you must provide parameters that identify your requested app client, scopes, redirect URI, and other details as query string parameters in HTTP GET requests. For IdP-initiated SAML assertions, however, the details of your request must be formatted as a `RelayState` parameter in the body of an HTTP POST request. The request body must also contain your SAML assertion as a `SAMLResponse` parameter.

The following is an example request for an IdP-initiated SAML provider.

```
POST /saml2/idpreponse HTTP/1.1
User-Agent: USER_AGENT
Accept: */*
Host: example.auth.us-east-1.amazoncognito.com
Content-Type: application/x-www-form-urlencoded
```

```
SAMLResponse=[Base64-encoded SAML assertion]&RelayState=identity_provider  
%3DMySAMLIdP%26client_id%3Dexample23456789%26redirect_uri%3Dhttps%3A%2F  
%2Fwww.example.com%26response_type%3Dcode%26scope%3Demail%2Bopenid%2Bphone
```

```
HTTP/1.1 302 Found  
Date: Wed, 06 Dec 2023 00:15:29 GMT  
Content-Length: 0  
x-amz-cognito-request-id: 8aba6eb5-fb54-4bc6-9368-c3878434f0fb  
Location: https://www.example.com?code=[Authorization code]
```

AWS Management Console

To configure an IdP for IdP-initiated SAML

1. Create a [user pool](#), [app client](#), and SAML identity provider.
2. Disassociate all social and OIDC identity providers from your app client, if any are associated.
3. Navigate to the **Social and external providers** menu of your user pool.
4. Edit or add a SAML provider.
5. Under **IdP-initiated SAML sign-in**, choose **Accept SP-initiated and IdP-initiated SAML assertions**.
6. Choose **Save changes**.

API/CLI

To configure an IdP for IdP-initiated SAML

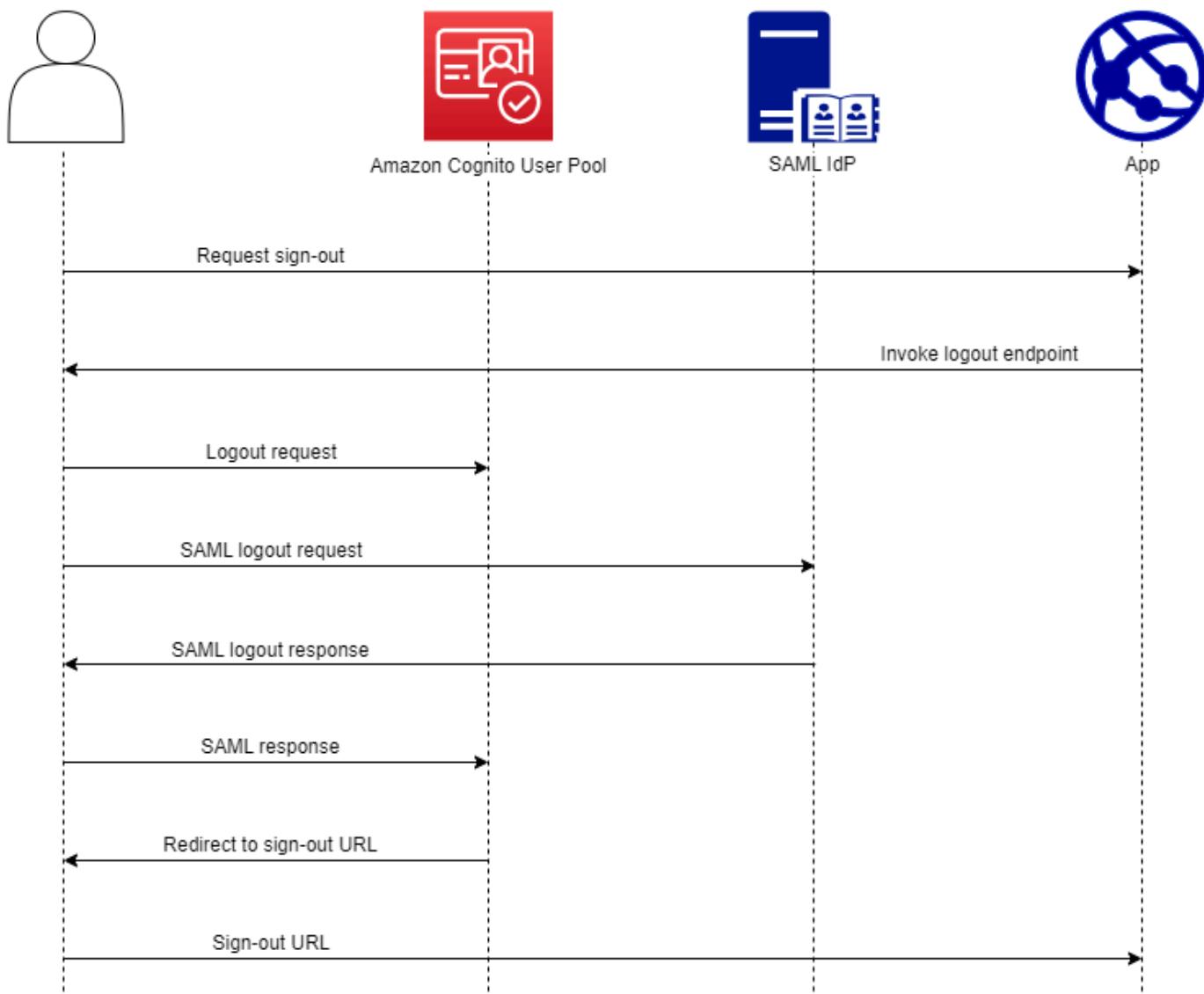
Configure IdP-initiated SAML with the `IDPInit` parameter in a [CreateIdentityProvider](#) or [UpdateIdentityProvider](#) API request. The following is an example `ProviderDetails` of an IdP that supports IdP-initiated SAML.

```
"ProviderDetails": {  
    "MetadataURL" : "https://myidp.example.com/saml/metadata",  
    "IDPSignout" : "true",  
    "RequestSigningAlgorithm" : "rsa-sha256",  
    "EncryptedResponses" : "true",  
    "IDPInit" : "true"  
}
```

Signing out SAML users with single sign-out

Amazon Cognito supports SAML 2.0 [single logout](#) (SLO). With SLO, your application can sign out users from their SAML identity providers (IdPs) when they sign out from your user pool. This way, when users want to sign in to your application again, they must authenticate with their SAML IdP. Otherwise, they might have IdP or user pool browser cookies in place that pass them through to your application without the requirement that they provide credentials.

When you configure your SAML IdP to support **Sign-out flow**, Amazon Cognito redirects your user with a signed SAML logout request to your IdP. Amazon Cognito determines the redirect location from the `SingleLogoutService URL` in your IdP metadata. Amazon Cognito signs the sign-out request with your user pool signing certificate.



When you direct a user with a SAML session to your user pool /logout endpoint, Amazon Cognito redirects your SAML user with the following request to the SLO endpoint that's specified in the IdP metadata.

```
https://[SingleLogoutService endpoint]?
SAMLRequest=[encoded SAML request]&
RelayState=[RelayState]&
SigAlg=http://www.w3.org/2001/04/xmldsig-more#rsa-sha256&
Signature=[User pool RSA signature]
```

Your user then returns to your saml2/logout endpoint with a LogoutResponse from their IdP. Your IdP must send the LogoutResponse in an HTTP POST request. Amazon Cognito then redirects them to the redirect destination from their initial sign-out request.

Your SAML provider might send a LogoutResponse with more than one AuthnStatement in it. The sessionIndex in the first AuthnStatement in a response of this type must match the sessionIndex in the SAML response that originally authenticated the user. If the sessionIndex is in any other AuthnStatement, Amazon Cognito won't recognize the session and your user won't be signed out.

AWS Management Console

To configure SAML sign-out

1. Create a [user pool](#), [app client](#), and SAML IdP.
2. When you create or edit your SAML identity provider, under **Identity provider information**, check the box with the title **Add sign-out flow**.
3. From the **Social and external providers** menu of your user pool, choose your IdP and locate the **Signing certificate**.
4. Choose **Download as .crt**.
5. Configure your SAML provider to support SAML single logout and request signing, and upload the user pool signing certificate. Your IdP must redirect to /saml2/logout in your user pool domain.

API/CLI

To configure SAML sign-out

Configure single logout with the `IDPSignout` parameter of a [CreateIdentityProvider](#) or [UpdateIdentityProvider](#) API request. The following is an example `ProviderDetails` of an IdP that supports SAML single logout.

```
"ProviderDetails": {  
    "MetadataURL" : "https://myidp.example.com/saml/metadata",  
    "IDPSignout" : "true",,  
    "RequestSigningAlgorithm" : "rsa-sha256",  
    "EncryptedResponses" : "true",  
    "IDPInit" : "true"  
}
```

SAML signing and encryption

SAML 2.0 sign-in is built around the user of an application as a bearer of requests and responses in their authentication flow. You might want to ensure that users aren't reading or modifying these SAML documents in transit. To accomplish this, add SAML signing and encryption to the SAML identity providers (IdPs) in your user pool. With SAML signing, your user pool adds a signature to SAML sign-in and sign-out requests. With your user pool public key, your IdP can verify that it's receiving unmodified SAML requests. Then, when your IdP responds and passes SAML assertions to users' browser sessions, the IdP can encrypt that response so that the user can't inspect their own attributes and entitlements.

With SAML signing and encryption, all cryptographic operations during user pool SAML operations must generate signatures and ciphertext with user-pool-provided keys that Amazon Cognito generates. Currently, you can't configure a user pool to sign requests or accept encrypted assertions with an external key.

Note

Your user pool certificates are valid for 10 years. Once per year, Amazon Cognito generates new signing and encryption certificates for your user pool. Amazon Cognito returns the most recent certificate when you request the signing certificate, and signs requests with the most recent signing certificate. Your IdP can encrypt SAML assertions with any user pool encryption certificate that isn't expired. Your previous certificates continue to be valid for their entire duration and the public key doesn't change between certificates. As a best practice, update the certificate in your provider configuration annually.

Topics

- [Accepting encrypted SAML responses from your IdP](#)
- [Signing SAML requests](#)

Accepting encrypted SAML responses from your IdP

Amazon Cognito and your IdP can establish confidentiality in SAML responses when users sign in and sign out. Amazon Cognito assigns a public-private RSA key pair and a certificate to each external SAML provider that you configure in your user pool. When you enable response encryption for your user pool SAML provider, you must upload your certificate to an IdP that supports encrypted SAML responses. Your user pool connection to your SAML IdP isn't functional before your IdP begins to encrypt all SAML assertions with the provided key.

The following is an overview of the flow of an encrypted SAML sign-in.

1. Your user starts sign-in and chooses their SAML IdP.
2. Your user pool [Authorize endpoint](#) redirects your user to their SAML IdP with a SAML sign-in request. Your user pool can optionally accompany this request with a signature that enables integrity verification by the IdP. When you want to sign SAML requests, you must configure your IdP to accept requests that your user pool has signed with the public key in the signing certificate.
3. The SAML IdP signs in your user and generates a SAML response. The IdP encrypts the response with the public key and redirects your user to your user pool /saml2/idpreponse endpoint. The IdP must encrypt the response as defined by the SAML 2.0 specification. For more information, see Element `<EncryptedAssertion>` in [Assertions and Protocols for the OASIS Security Assertion Markup Language \(SAML\) V2.0](#).
4. Your user pool decrypts the ciphertext in the SAML response with the private key and signs in your user.

Important

When you enable response encryption for a SAML IdP in your user pool, your IdP must encrypt all responses with a public key that's specific to the provider. Amazon Cognito doesn't accept unencrypted SAML responses from a SAML external IdP that you configure to support encryption.

Any external SAML IdP in your user pool can support response encryption, and each IdP receives its own key pair.

AWS Management Console

To configure SAML response encryption

1. Create a [user pool](#), [app client](#), and SAML IdP.
2. When you create or edit your SAML identity provider, under **Sign requests and encrypt responses**, check the box with the title **Require encrypted SAML assertions from this provider**.
3. From the **Social and external providers** menu of your user pool, select your SAML IdP and choose **View encryption certificate**.
4. Choose **Download as .crt** and provide the downloaded file to your SAML IdP. Configure your SAML IdP to encrypt SAML responses with the key in the certificate.

API/CLI

To configure SAML response encryption

Configure response encryption with the `EncryptedResponses` parameter of a [CreateIdentityProvider](#) or [UpdateIdentityProvider](#) API request. The following is an example `ProviderDetails` of an IdP that supports request signing.

```
"ProviderDetails": {  
    "MetadataURL" : "https://myidp.example.com/saml/metadata",  
    "IDPSignout" : "true",  
    "RequestSigningAlgorithm" : "rsa-sha256",  
    "EncryptedResponses" : "true",  
    "IDPInit" : "true"  
}
```

To get the encryption certificate from your user pool, make a [DescribeIdentityProvider](#) API request and retrieve the value of `ActiveEncryptionCertificate` in the response parameter `ProviderDetails`. Save this certificate and provide it to your IdP as the encryption certificate for sign-in requests from your user pool.

Signing SAML requests

The ability to prove the integrity of SAML 2.0 requests to your IdP is a security advantage of Amazon Cognito SP-initiated SAML sign-in. Each user pool with a domain receives a user pool X.509 signing certificate. With the public key in this certificate, user pools apply a cryptographic signature to the *sign-out requests* that your user pool generates when your users select a SAML IdP. You can optionally configure your app client to sign SAML *sign-in requests*. When you sign your SAML requests, your IdP can check that the signature in the XML metadata of your requests matches the public key in the user pool certificate that you provide.

AWS Management Console

To configure SAML request signing

1. Create a [user pool](#), [app client](#), and SAML IdP.
2. When you create or edit your SAML identity provider, under **Sign requests and encrypt responses**, check the box with the title **Sign SAML requests to this provider**.
3. From the **Social and external providers** menu of your user pool, choose **View signing certificate**.
4. Choose **Download as .crt** and provide the downloaded file to your SAML IdP. Configure your SAML IdP to verify the signature of incoming SAML requests.

API/CLI

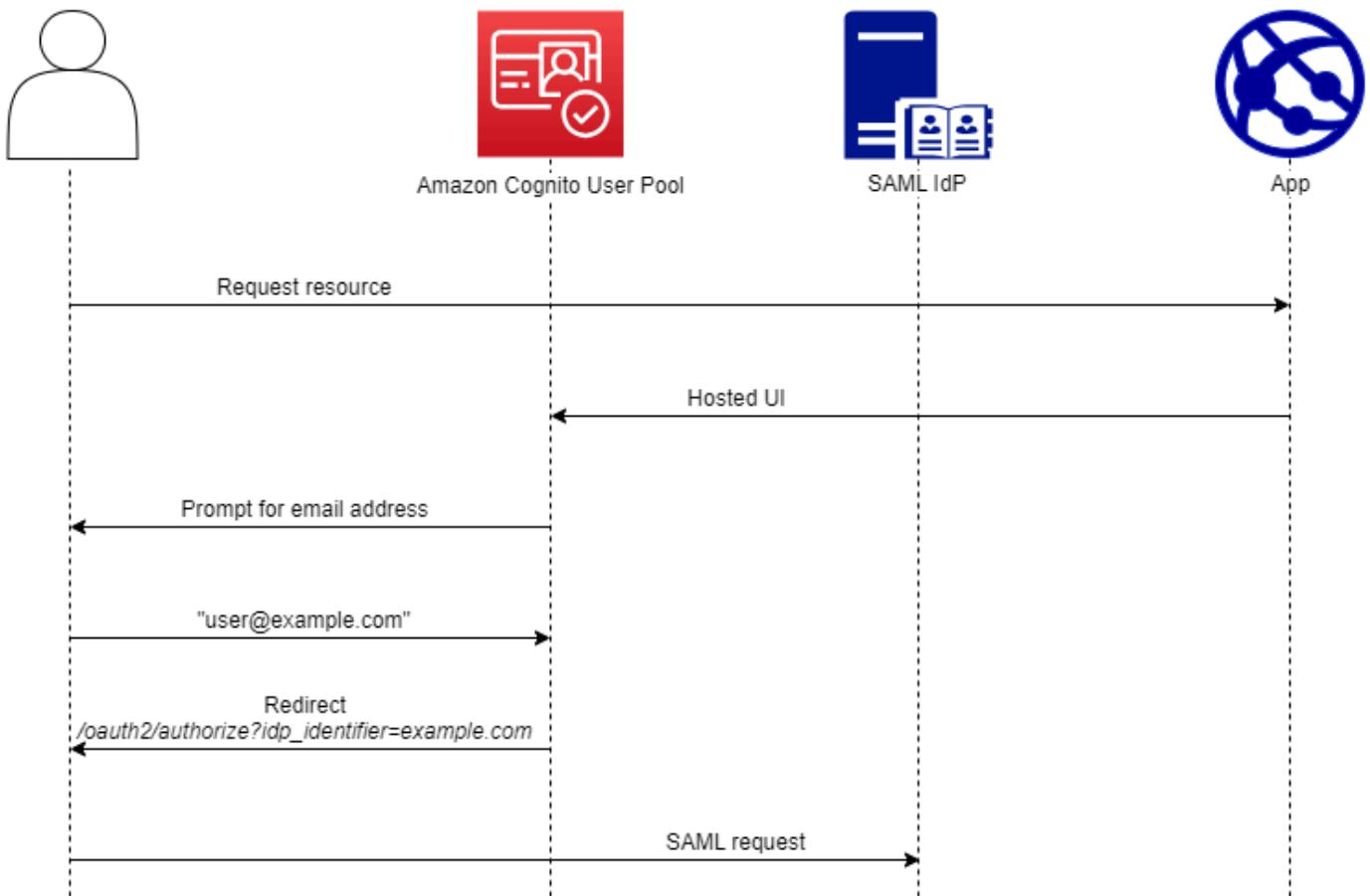
To configure SAML request signing

Configure request signing with the `RequestSigningAlgorithm` parameter of a [CreateIdentityProvider](#) or [UpdateIdentityProvider](#) API request. The following is an example `ProviderDetails` of an IdP that supports request signing.

```
"ProviderDetails": {  
    "MetadataURL" : "https://myidp.example.com/saml/metadata",  
    "IDPSignout" : "true",  
    "RequestSigningAlgorithm" : "rsa-sha256",  
    "EncryptedResponses" : "true",  
    "IDPInit" : "true"  
}
```

SAML identity provider names and identifiers

When you name your SAML identity providers (IdPs) and assign IdP identifiers, you can automate the flow of SP-initiated sign-in and sign-out requests to that provider. For information about string constraints to the provider name, see the `ProviderName` property of [CreateIdentityProvider](#).



You can also choose up to 50 identifiers for your SAML providers. An identifier is a friendly name for an IdP in your user pool, and must be unique within the user pool. If your SAML identifiers match your users' email domains, managed login requests each user's email address, evaluates the domain in their email address, and redirects them to the IdP that corresponds to their domain. Because the same organization can own multiple domains, a single IdP can have multiple identifiers.

Whether you use or don't use email-domain identifiers, you can use identifiers in a multi-tenant app to redirect users to the correct IdP. When you want to bypass managed login entirely, you can customize the links that you present to users such that they redirect through the [Authorize endpoint](#) directly to their IdP. To sign in your users with an identifier and redirect to their IdP,

include the identifier in the format `idp_identifier=`*myidp.example.com* in the request parameters of their initial authorization request.

Another method to pass a user through to your IdP is to populate the parameter `identity_provider` with the name of your IdP in the following URL format.

```
https://mydomain.us-east-1.amazonaws.com/oauth2/authorize?  
response_type=code&  
identity_provider=MySAMLIdP&  
client_id=lexample23456789&  
redirect_uri=https://www.example.com
```

After a user signs in with your SAML IdP, your IdP redirects them with a SAML response in the HTTP POST body to your `/saml2/idpresponse` endpoint. Amazon Cognito processes the SAML assertion and, if the claims in the response meet expectations, redirects to your app client callback URL. After your user has completed authentication in this way, they have interacted with webpages for only your IdP and your app.

With IdP identifiers in a domain format, managed login requests email addresses at sign-in and then, when the email domain matches an IdP identifier, redirects users to the sign-in page for their IdP. As an example, you build an app that requires sign-in by employees of two different companies. The first company, AnyCompany A, owns `exampleA.com` and `exampleA.co.uk`. The second company, AnyCompany B, owns `exampleB.com`. For this example, you have set up two IdPs, one for each company, as follows:

- For IdP A, you define identifiers `exampleA.com` and `exampleA.co.uk`.
- For IdP B, you define identifier `exampleB.com`.

In your app, invoke managed login for your app client to prompt each user to enter their email address. Amazon Cognito derives the domain from the email address, correlates the domain to an IdP with a domain identifier, and redirects your user to the correct IdP with a request to the [Authorize endpoint](#) that contains an `idp_identifier` request parameter. For example, if a user enters `bob@exampleA.co.uk`, the next page that they interact with is the IdP sign-in page at `https://auth.exampleA.co.uk/sso/saml`.

You can also implement the same logic independently. In your app, you can build a custom form that collects user input and correlates it to the correct IdP according to your own logic. You can

generate custom portals for each of your app tenants, where each links to the authorize endpoint with the tenant's identifier in the request parameters.

To collect an email address and parse the domain in managed login, assign at least one identifier to each SAML IdP that you have assigned to your app client. By default, the managed login sign-in screen displays a button for each of the IdPs that you have assigned to your app client. However, if you have successfully assigned identifiers, your managed login sign-in page looks like the following image.

Domain parsing in managed login requires that you use domains as your IdP identifiers. If you assign an identifier of any type to each of the SAML IdPs for an app client, managed login for that app no longer displays IdP-selection buttons. Add IdP identifiers for SAML when you intend to use email parsing or custom logic to generate redirects. When you want to generate silent redirects and also want your managed login pages to display a list of IdPs, don't assign identifiers and use the `identity_provider` request parameter in your authorization requests.

- If you assign only one SAML IdP to your app client, the managed login sign-in page displays a button to sign in with that IdP.
- If you assign an identifier to every SAML IdP that you activate for your app client, a user input prompt for an email address appears in the managed login sign-in page.
- If you have multiple IdPs and you do not assign an identifier to all of them, the managed login sign-in page displays a button to sign in with each assigned IdP.
- If you assigned identifiers to your IdPs and you want your managed login pages to display a selection of IdP buttons, add a new IdP that has no identifier to your app client, or create a new app client. You can also delete an existing IdP and add it again without an identifier. If you create a new IdP, your SAML users will create new user profiles. This duplication of active users might have a billing impact in the month that you change your IdP configuration.

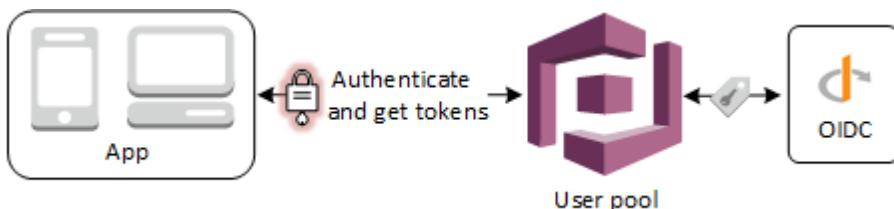
For more information about IdP setup, see [Configuring identity providers for your user pool](#).

Using OIDC identity providers with a user pool

Users can sign in to your application using their existing accounts from OpenID Connect (OIDC) identity providers (IdPs). With OIDC providers, users of independent single sign-on systems can provide existing credentials while your application receives OIDC tokens in the shared format of user pools. A user pool is an OIDC IdP that can also serve as an intermediate relying party between multiple external OIDC IdPs and your application.

Users who sign in with an OIDC IdP aren't required to provide new credentials or information to access your user pool application. Your application can silently redirect them to their IdP for sign-in, with a user pool as a tool in the background that standardizes the token format for your application. To learn more about IdP redirection, see [Authorize endpoint](#).

Like with other third-party identity providers, you must register your application with the OIDC provider and obtain information about the IdP application that you want to connect to your user pool. A user pool OIDC IdP requires a client ID, client secret, scopes that you want to request, and information about provider service endpoints. Your user pool can discover the provider OIDC endpoints from a discovery endpoint or you can enter them manually. You must also examine provider ID tokens and create attribute mappings between the IdP and the attributes in your user pool.



Note

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

You can add an OIDC IdP to your user pool in the AWS Management Console, through the AWS CLI, or with the user pool API method [CreateIdentityProvider](#).

Topics

- [Prerequisites](#)
- [Step 1: Register with an OIDC IdP](#)
- [Step 2: Add an OIDC IdP to your user pool](#)
- [Step 3: Test your OIDC IdP configuration](#)
- [OIDC user pool IdP authentication flow](#)

Prerequisites

Before you begin, you need the following:

- A user pool with an app client and a user pool domain. For more information, see [Create a user pool](#).
- An OIDC IdP with the following configuration:
 - Supports `client_secret_post` client authentication. Amazon Cognito doesn't check the `token_endpoint_auth_methods_supported` claim at the OIDC discovery endpoint for your IdP. Amazon Cognito doesn't support `client_secret_basic` client authentication. For more information on client authentication, see [Client Authentication](#) in the OpenID Connect documentation.
 - Only uses HTTPS for OIDC endpoints such as `openid_configuration`, `userInfo`, and `jwks_uri`.
 - Only uses TCP ports 80 and 443 for OIDC endpoints.
 - Only signs ID tokens with HMAC-SHA, ECDSA, or RSA algorithms.
 - Publishes a key ID `kid` claim at its `jwks_uri` and includes a `kid` claim in its tokens.
 - Presents a non-expired public key with a valid root CA trust chain.

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://<i>Your Ping domain address</i>:9031/idp/userinfo openid

OIDC IdP	How to Install	OIDC Discovery URL
		For example: <code>https://pf.company.com:9031/idp/userinfo.openid</code>
Okta	Install an Okta identity provider	<code>https://Your Okta subdomain.oktapreview.com</code> or <code>https://Your Okta subdomain.okta.com</code>
Microsoft Azure Active Directory (Azure AD)	Install a Microsoft Azure AD identity provider	<code>https://login.microsoftonline.com/ {tenant}/v2.0</code>
Google	Install a Google identity provider	<code>https://accounts.google.com</code> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> Note Amazon Cognito offers Google as an integrated social sign-in IdP. We recommend that you use the integrated IdP. See Using social identity providers with a user pool. </div>

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://mydomain.us-east-1.amazoncognito.com/oauth2/idpresponse`

3. Register your callback URL with your Amazon Cognito user pool. This is the URL of the page where Amazon Cognito redirects your user 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 IdP 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. From your Salesforce page, do one of the following:
 - 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**, enter a domain, and choose **Create**.
6. On the left navigation bar, under **Platform Tools**, choose **Apps**.
7. Choose **App Manager**.
8. a. Choose **New connected app**.
b. Complete the required fields.

Under **Start URL**, enter a URL at the /authorize endpoint for the user pool domain that signs in with your Salesforce IdP. When your users access your connected app, Salesforce directs them to this URL to complete sign-in. Then Salesforce redirects the users to the callback URL that you have associated with your app client.

```
https://mydomain.us-east-1.amazonaws.com/authorize?  
response_type=code&client_id=<your_client_id>&redirect_uri=https://  
www.example.com&identity_provider=CorpSalesforce
```

- c. Enable **OAuth settings** and enter the URL of the /oauth2/idresponse endpoint for your user pool domain in **Callback URL**. This is the URL where Salesforce issues the authorization code that Amazon Cognito exchanges for an OAuth token.

`https://mydomain.us-east-1.amazoncognito.com/oauth2/idresponse`

9. Select your scopes. You must include the scope **openid**. To grant access to the **email** and **email_verified** claims, add the **email** scope. Separate scopes 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)

Add an OIDC IdP

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools** from the navigation menu.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Social and external providers** menu and then select **Add an identity provider**.
5. Choose an **OpenID Connect IdP**.
6. Enter a unique name into **Provider name**.
7. Enter the client ID you received from your provider into **Client ID**.
8. Enter the client secret you received from your provider into **Client secret**.
9. Enter **Authorized scopes** for this provider. Scopes define which groups of user attributes (such as name and email) that your application will request from your provider. Scopes must be separated by spaces, following the [OAuth 2.0](#) specification.

Your user will be asked to consent to providing these attributes to your application.

10. Choose an **Attribute request method** to provide Amazon Cognito with the HTTP method (either GET or POST) that it must use to fetch the details of the user from the **userInfo** endpoint operated by your provider.
11. Choose a **Setup method** to retrieve OpenID Connect endpoints either by **Auto fill through issuer URL** or **Manual input**. Use **Auto fill through issuer URL** when your provider has a public `.well-known/openid-configuration` endpoint where Amazon Cognito can retrieve the URLs of the authorization, token, userInfo, and jwks_uri endpoints.
12. Enter the issuer URL or authorization, token, userInfo, and jwks_uri endpoint URLs from your IdP.

 **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 you choose auto fill, the discovery document must use HTTPS for the following values: `authorization_endpoint`, `token_endpoint`, `userinfo_endpoint`, and `jwks_uri`. Otherwise the login will fail.

13. The OIDC claim **sub** is mapped to the user pool attribute **Username** by default. You can map other OIDC [claims](#) to user pool attributes. Enter 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**.
14. Map attributes from your IdP to your user pool. For more information, see [Specifying Identity Provider Attribute Mappings for Your User Pool](#).
15. Choose **Create**.
16. From the **App clients** menu, select an app client from the list and select **Edit**. Add the new OIDC identity provider to the app client under **Identity providers**.
17. Choose **Save changes**.

To add an OIDC IdP (AWS CLI)

- See the parameter descriptions for the [CreateIdentityProvider](#) API method.

```
aws cognito-identity 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://mydomain.us-east-1.amazoncognito.com/oauth2/authorize?  
response\_type=code&client\_id=1example23456789&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.

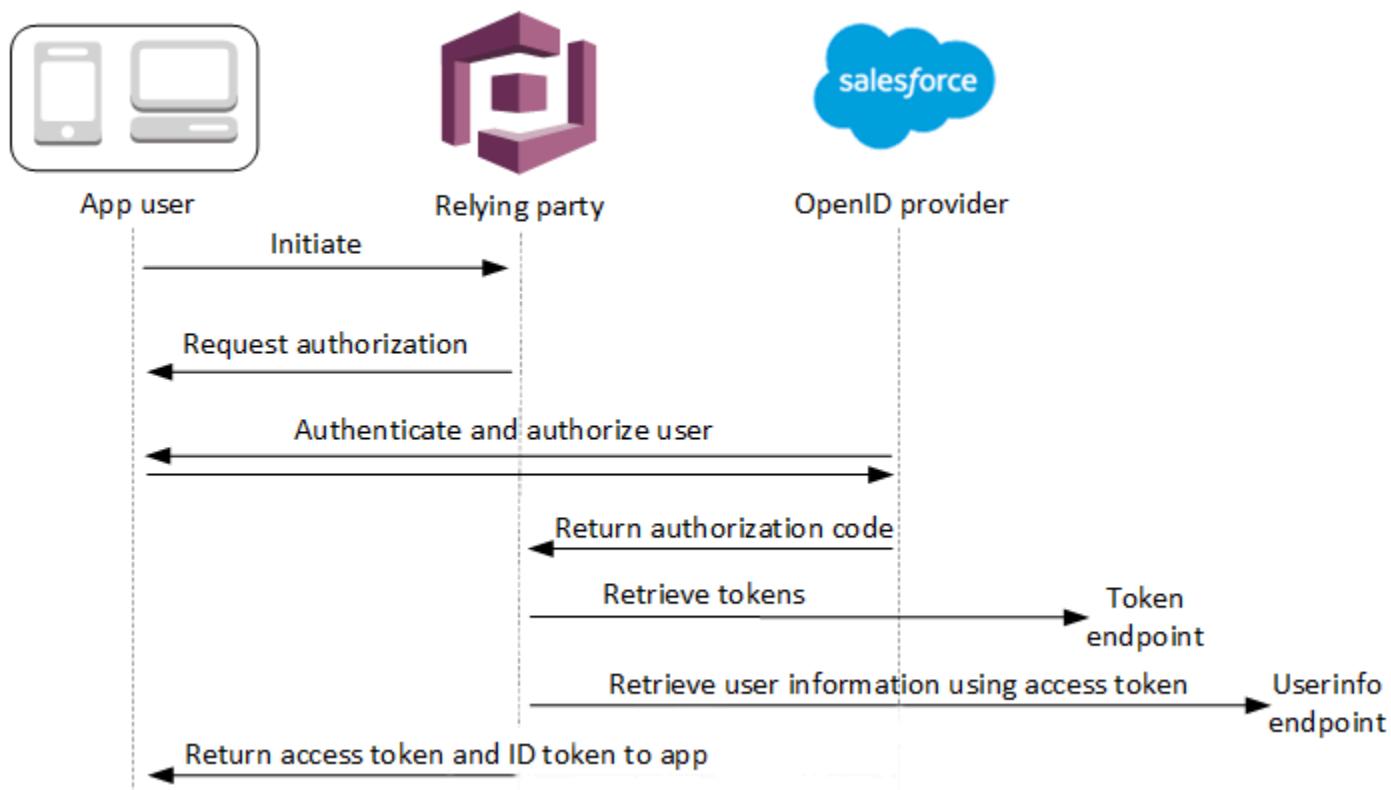
OIDC user pool IdP authentication flow

With OpenID Connect (OIDC) sign-in, your user pool automates an authorization-code sign-in flow with your identity provider (IdP). After your user completes sign-in with their IdP, Amazon Cognito collects their code at the `oauth2/idpresponse` endpoint of the external provider. With the resulting access token, your user pool queries the IdP `userInfo` endpoint to retrieve user attributes. Your user pool then compares the received attributes to the attribute-mapping rules you've set up and populates the user's profile and ID token accordingly.

The OAuth 2.0 scopes that you request in your OIDC provider configuration define the user attributes that the IdP provides to Amazon Cognito. As a best security practice, only request the scopes that correspond to attributes that you want to map to your user pool. For example, if your user pool requests `openid profile`, you'll get all possible attributes, but if you request `openid email phone_number` you'll only get the user's email address and phone number. You can configure the scopes that you [request from OIDC IdPs](#) to differ from those you authorize and request in the [app client](#) and user pool authentication request.

When your user signs in to your application using an OIDC IdP, your user pool conducts the following 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, for example Salesforce.
2. Your user is redirected to the authorization endpoint of the OIDC IdP.
3. After your user authenticates, 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.



Note

Amazon Cognito cancels authentication requests that do not complete within 5 minutes, and redirects the user to managed login. The page displays 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 with an authorization code grant, 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. For more information about grant types that your user pool app client can support, see [Authorize endpoint](#).

How a user pool processes claims from an OIDC provider

When your user completes sign-in with a third-party OIDC provider, managed login retrieves an authorization code from the IdP. Your user pool exchanges the authorization code for access and ID

tokens with the token endpoint of your IdP. Your user pool doesn't pass these tokens on to your user or your app, but uses them to build a user profile with data that it presents in claims in its own tokens.

Amazon Cognito doesn't independently validate the access token. Instead, it requests user-attribute information from the provider `userInfo` endpoint and expects the request to be denied if the token isn't valid.

Amazon Cognito validates the provider ID token with the following checks:

1. Check that the provider signed the token with an algorithm from the following set: RSA, HMAC, Elliptic Curve.
2. If the provider signed the token with an asymmetric signing algorithm, check that the signing key ID in the token `kid` claim is listed at the provider `jwks_uri` endpoint.
3. Compare the ID token signature to the signature that it expects based on provider metadata.
4. Compare the `iss` claim to the OIDC issuer configured for the IdP.
5. Compare the `aud` claim matches the client ID configured on the IdP, or that it contains the configured client ID if there are multiple values in the `aud` claim.
6. Check that the timestamp in the `exp` claim is not before the current time.

Your user pool validates the ID token, then attempts a request to the provider `userInfo` endpoint with the provider access token. It retrieves any user profile information that the scopes in the access token authorize it to read. Your user pool then searches for the user attributes that you have set as required in your user pool. You must create attribute mappings in your provider configuration for required attributes. Your user pool checks the provider ID token and the `userInfo` response. Your user pool writes all claims that match mapping rules to user attributes on the user pool user profile. Your user pool ignores attributes that match a mapping rule but aren't required and aren't found in the provider's claims.

Mapping IdP attributes to profiles and tokens

Identity provider (IdP) services, including Amazon Cognito, can typically record more information about a user. You might want to know what company they work for, how to contact them, and other identifying information. But the format that these attributes take has variations between providers. For example, set up three IdPs from three different vendors with your user pool and examine an example SAML assertion, ID token, or `userInfo` payload from each. One will

represent the user's email address as `email`, another as `emailaddress`, and the third as `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`.

A major benefit that comes from consolidation of IdPs with a user pool is the ability to map the variety of attribute names into a single OIDC token schema with consistent, predictable, shared attribute names. This way, your developers aren't required to maintain the logic for processing a complex variety of single sign-on events. This format consolidation is attribute mapping. User pool attribute mapping assigns IdP attribute names to the corresponding user pool attribute names. For example, you can configure your user pool to write the value of an `emailaddress` claim to the standard user pool attribute `email`.

Each user pool IdP has a separate attribute-mapping schema. To specify attribute mappings for your IdP, configure a user pool identity provider in the Amazon Cognito console, an AWS SDK, or the user pools REST API .

Things to know about mappings

Before you begin to set up user-attribute mapping, review the following important details.

- When a federated user signs in to your application, a mapping must be present for each user pool attribute that your user pool requires. For example, if your user pool requires an `email` attribute for sign-up, map this attribute to its equivalent from the IdP.
- 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 IdP to get the verification status. For example, Google and most OIDC providers include the `email_verified` attribute.
- You can map identity provider (IdP) tokens to custom attributes in your user pool. Social providers present an access token, and OIDC providers present an access and ID token. To map a token, add a custom attribute with a maximum length of 2,048 characters, grant your app client write access to the attribute, and map `access_token` or `id_token` from the IdP to the custom attribute.
- For each mapped user pool attribute, the maximum value length of 2,048 characters must be large enough for the value that Amazon Cognito obtains from the IdP. Otherwise, Amazon Cognito reports an error when users sign in to your application. Amazon Cognito doesn't support mapping IdP tokens to custom attributes when the tokens are more than 2,048 characters long.
- Amazon Cognito derives the `username` attribute in a federated user's profile from specific claims that your federated IdP passes, as shown in the following table. Amazon Cognito prepends this attribute value with the name of your IdP, for example `MyOIDCIdP_[sub]`. When you want

your federated users to have an attribute that exactly matches an attribute in your external user directory, map that attribute to a Amazon Cognito sign-in attribute like `preferred_username`.

Identity Provider	username source attribute
Facebook	<code>id</code>
Google	<code>sub</code>
Login with Amazon	<code>user_id</code>
Sign in with Apple	<code>sub</code>
SAML providers	<code>NameID</code>
OpenID Connect (OIDC) providers	<code>sub</code>

- When a user pool is [case insensitive](#), Amazon Cognito converts the username source attribute to lowercase in federated users' automatically-generated usernames. The following is an example username for a case-sensitive user pool: `MySAML_TestUser@example.com`. The following is the same username for a *case-insensitive* user pool: `MySAML_testuser@example.com`.

In case-insensitive user pools, your Lambda triggers that process the username must account for this modification to any mixed-case claims for user name source attributes. To link your IdP to a user pool that has a different case-sensitivity setting than your current user pool, create a new user pool.

- 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 IdP, Amazon Cognito updates the mapped attributes with the latest information from the IdP. Amazon Cognito updates each mapped attribute, even if its current value already matches the latest information. To ensure that Amazon Cognito can update the attributes, check the following requirements:
 - All of the user pool custom attributes that you map from your IdP must be *mutable*. You can update mutable custom attributes at any time. By contrast, you can only set a value for a user's *immutable* custom attribute when you first create the user profile. To create a mutable custom attribute in the Amazon Cognito console, activate the **Mutable** checkbox for the attribute you add when you select **Add custom attributes** in the **Sign-up** menu. 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`. If your IdP sends a value for a mapped immutable attribute, Amazon Cognito returns an error and sign-in fails.

- 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. If your IdP sends a value for a mapped non-writable attribute, Amazon Cognito doesn't set the attribute value and proceeds with authentication.
- When IdP attributes contain multiple values, Amazon Cognito flattens all values into a single comma-delimited string and URL form-encodes the values containing non-alphanumeric characters (excluding the '.', '-', '*', and '_' characters). You must decode and parse the individual values before you use them in your app.

Specifying identity provider attribute mappings for your user pool (AWS Management Console)

You can use the AWS Management Console to specify attribute mappings for the IdP your user pool.

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 repopulate from the incoming token.

To specify a social IdP attribute mapping

1. Sign in to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Choose **Add an identity provider**, or choose the **Facebook, Google, Amazon or Apple** IdP you have configured. Locate **Attribute mapping** and choose **Edit**.

For more information about adding a social IdP, see [Using social identity providers with a user pool](#).

5. For each attribute you need to map, complete the following steps:
 - a. Select an attribute from the **User pool attribute** column. This is the attribute that is assigned to the user profile in your user pool. Custom attributes are listed after standard attributes.
 - b. Select an attribute from the **<provider> attribute** column. This will be the attribute passed from the provider directory. Known attributes from the social provider are provided in a drop-down list.
 - c. To map additional attributes between your IdP and Amazon Cognito, choose **Add another attribute**.
6. Choose **Save changes**.

To specify a SAML provider attribute mapping

1. Sign in to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Choose **Add an identity provider**, or choose the SAML IdP you have configured. Locate **Attribute mapping**, and choose **Edit**. For more information about adding a SAML IdP, see [Using SAML identity providers with a user pool](#).
5. For each attribute you need to map, complete the following steps:
 - a. Select an attribute from the **User pool attribute** column. This is the attribute that is assigned to the user profile in your user pool. Custom attributes are listed after standard attributes.
 - b. Select an attribute from the **SAML attribute** column. This will be the attribute passed from the provider directory.

Your IdP might offer sample SAML assertions for reference. Some IdPs use simple names, such as `email`, while others use URL-formatted attribute names similar to:

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

- c. To map additional attributes between your IdP and Amazon Cognito, choose **Add another attribute**.
6. Choose **Save changes**.

Specifying identity provider attribute mappings for your user pool (AWS CLI and AWS API)

The following request body for [CreateIdentityProvider](#) or [UpdateIdentityProvider](#) maps the SAML provider "MyIdP" attributes emailaddress, birthdate, and phone to the user pool attributes email, birthdate, and phone_number, in that order. This is a complete request body for a SAML 2.0 provider—your request body will vary depending on IdP type and specific details. The attribute mapping is in the AttributeMapping parameter.

```
{  
    "AttributeMapping": {  
        "email" : "emailaddress",  
        "birthdate" : "birthdate",  
        "phone_number" : "phone"  
    },  
    "IdpIdentifiers": [  
        "IdP1",  
        "pdxsaml"  
    ],  
    "ProviderDetails": {  
        "IDPInit": "true",  
        "IDPSignout": "true",  
        "EncryptedResponses" : "true",  
        "MetadataURL": "https://auth.example.com/sso/saml/metadata",  
        "RequestSigningAlgorithm": "rsa-sha256"  
    },  
    "ProviderName": "MyIdP",  
    "ProviderType": "SAML",  
    "UserPoolId": "us-west-2_EXAMPLE"  
}
```

Use the following commands to specify IdP attribute mappings for your user pool.

To specify attribute mappings at provider creation time

- AWS CLI: `aws cognito-identity-provider 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 us-east-1_EXAMPLE \  
--provider-name=SAML_provider_1 \  
--provider-type SAML \  
--provider-details MetadataURL=https://myidp.example.com/saml/metadata \  
--attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/\  
emailaddress
```

- API/SDK: [CreateIdentityProvider](#)

To specify attribute mappings for an existing IdP

- 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

- API/SDK: [UpdateIdentityProvider](#)

To get information about attribute mapping for a specific IdP

- 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*>

- API/SDK: [DescribeldentityProvider](#)

Linking federated users to an existing user profile

Often, the same user has a profile with multiple identity providers (IdPs) that you have connected to your user pool. Amazon Cognito can link each occurrence of a user to the same user profile in your directory. This way, one person who has multiple IdP users can have a consistent experience in your app. [AdminLinkProviderForUser](#) tells Amazon Cognito to recognize a user's unique ID in your federated directory as a user in the user pool. A user in your user pool counts as one monthly active user (MAU) for the purposes of [billing](#) when you have zero or more federated identities associated with the user profile.

When a federated user signs in to your user pool for the first time, Amazon Cognito looks for a local profile that you have linked to their identity. If no linked profile exists, your user pool creates a new profile. You can create a local profile and link it to your federated user at any time before their first sign-in, in an AdminLinkProviderForUser API request, either in a planned prestaging task or in a [Pre sign-up Lambda trigger](#). After your user signs in and Amazon Cognito detects a linked local profile, your user pool reads your user's claims and compares them to mapping rules for the IdP. Your user pool then updates the linked local profile with the claims mapped from their sign-in. In this way, you can configure the local profile with access claims and keep their identity claims up-to-date with your provider. After Amazon Cognito matches your federated user to a linked profile, they always sign in to that profile. You can then link more of your user's provider identities to the same profile, giving one customer a consistent experience in your app. To link a federated user who has previously signed in, you must first delete their existing profile. You can identify existing profiles by their format: [*Provider name*]_identifier. For example, LoginWithAmazon_amzn1.account.AFAEXAMPLE. A user that you created and then linked to a third-party user identity has the username that they were created with, and an identities attribute that contains the details of their linked identities.

Important

Because `AdminLinkProviderForUser` allows a user with an external federated identity to sign in as an existing user in the user pool, it is critical that it only be used with external IdPs and provider attributes that have been trusted by the application owner.

For example, if you're a managed service provider (MSP) with an app that you share with multiple customers. Each of the customers signs in to your app through Active Directory Federation Services (ADFS). Your IT administrator, Carlos, has an account in each of your customers' domains. You want Carlos to be recognized as an app administrator every time he signs in, regardless of the IdP.

Your ADFS IdPs present Carlos' email address `msp_carlos@example.com` in the `email` claim of the Carlos' SAML assertions to Amazon Cognito. You create a user in your user pool with the user name Carlos. The following AWS Command Line Interface (AWS CLI) commands link Carlos' identities from IdPs ADFS1, ADFS2, and ADFS3.

Note

You can link a user based on specific attribute claims. This ability is unique to OIDC and SAML IdPs. For other provider types, you must link based on a fixed source attribute. For more information, see [AdminLinkProviderForUser](#). You must set `ProviderAttributeName` to `Cognito_Subject` when you link a social IdP to a user profile. `ProviderAttributeValue` must be the user's unique identifier with your IdP.

```
aws cognito-idp admin-link-provider-for-user \
--user-pool-id us-east-1_EXAMPLE \
--destination-user ProviderAttributeValue=Carlos,ProviderName=Cognito \
--source-user
  ProviderName=ADFS1,ProviderAttributeName=email,ProviderAttributeValue=msp_carlos@example.com

aws cognito-idp admin-link-provider-for-user \
--user-pool-id us-east-1_EXAMPLE \
--destination-user ProviderAttributeValue=Carlos,ProviderName=Cognito \
--source-user
  ProviderName=ADFS2,ProviderAttributeName=email,ProviderAttributeValue=msp_carlos@example.com

aws cognito-idp admin-link-provider-for-user \
```

```
--user-pool-id us-east-1_EXAMPLE \
--destination-user ProviderAttributeValue=Carlos,ProviderName=Cognito \
--source-user
  ProviderName=ADFS3,ProviderAttributeName=email,ProviderAttributeValue=msp_carlos@example.com
```

The user profile Carlos in your user pool now has the following identities attribute.

```
[{
  "userId": "msp_carlos@example.com",
  "providerName": "ADFS1",
  "providerType": "SAML",
  "issuer": "http://auth.example.com",
  "primary": false,
  "dateCreated": 1111111111111111
}, {
  "userId": "msp_carlos@example.com",
  "providerName": "ADFS2",
  "providerType": "SAML",
  "issuer": "http://auth2.example.com",
  "primary": false,
  "dateCreated": 1111111111111111
}, {
  "userId": "msp_carlos@example.com",
  "providerName": "ADFS3",
  "providerType": "SAML",
  "issuer": "http://auth3.example.com",
  "primary": false,
  "dateCreated": 1111111111111111
}]
```

Things to know about linking federated users

- You can link up to five federated users to each user profile.
- You can link users to each IdP from up to five IdP attribute claims, as defined by the `ProviderAttributeName` parameter of `SourceUser` in an `AdminLinkProviderForUser` API request. For example, if you have linked at least one user to the source attributes `email`, `phone`, `department`, `given_name`, and `location`, you can only link additional users on one of those five attributes.
- You can link federated users to either an existing federated user profile, or to a local user.
- You can't link providers to user profiles in the AWS Management Console.

- Your user's ID token contains all of their associated providers in the `identities` claim.
- You can set a password for the automatically-created federated user profile in an [AdminSetUserPassword](#) API request. That user's status then changes from EXTERNAL_PROVIDER to CONFIRMED. A user in this state can sign in as a federated user, and initiate authentication flows in the API like a linked local user. They can also modify their password and attributes in token-authenticated API requests like [ChangePassword](#) and [UpdateUserAttributes](#). As a best security practice and to keep users in sync with your external IdP, don't set passwords on federated user profiles. Instead, link users to local profiles with [AdminLinkProviderForUser](#).
- Amazon Cognito populates user attributes to a linked local user profile when the user signs in through their IdP. Amazon Cognito processes identity claims in the ID token from an OIDC IdP, and also checks the `userInfo` endpoint of both OAuth 2.0 and OIDC providers. Amazon Cognito prioritizes information in an ID token over information from `userInfo`.

When you learn that your user is no longer using an external user account that you've linked to their profile, you can disassociate that user account with your user pool user. When you linked your user, you supplied the user's attribute name, attribute value and provider name in the request. To remove a profile that your user no longer needs, make an [AdminDisableProviderForUser](#) API request with equivalent parameters.

See [AdminLinkProviderForUser](#) for additional command syntax and examples in the AWS SDKs.

User pool managed login

You can choose a web domain to host services for your user pool. An Amazon Cognito user pool gains the following functions when you add a domain, collectively referred to as *managed login*.

- An *authorization server* that acts as an identity provider (IdP) to applications that work with OAuth 2.0 and OpenID Connect (OIDC). The authorization server routes authentication requests, issues and manages JSON web tokens (JWTs), and delivers user attribute information.
- A ready-to-use user interface (UI) for authentication operations like sign-in, sign-out and password management. The *managed login pages* act as a web front end for authentication services.
- A service provider (SP), or relying party (RP), to SAML 2.0 IdPs, OIDC IdPs, Facebook, Login with Amazon, Sign in with Apple, and Google.

An additional option that shares some features with managed login is the classic *hosted UI*. The classic hosted UI is a first-generation version of the managed login services. Hosted UI IdP and RP services generally have the same characteristics as managed login, but the login pages have a simpler design and fewer features. For example, passkey sign-in isn't available in the classic hosted UI. In the Lite [feature plan](#), the classic hosted UI is your only option for user pool domain services.

The managed login pages are a collection of web interfaces for basic sign-up, sign-in, multi-factor authentication and password-reset activities in your user pool. They also connect users to one or more third-party identity providers (IdPs) when you want to give users a choice of sign-in option. Your app can invoke your managed login pages in users' browsers when you want to authenticate and authorize users.

You can make the managed login user experience fit your brand with custom logos, backgrounds, styles, and UI text. You have two options for the branding that you might apply to your managed login UI: the *branding designer* for managed login, and *hosted UI (classic) branding* for the hosted UI.

Branding designer

An updated user experience with the most up-to-date authentication options and a visual editor in the Amazon Cognito console.

Hosted UI branding

A familiar user experience for previous adopters of Amazon Cognito user pools. Branding for the hosted UI is a file-based system. To apply branding to hosted UI pages, you upload a logo image file and a file that sets values for several predetermined CSS style options.

The branding designer isn't available in all feature plans for user pools. For more information, see [User pool feature plans](#).

For more information about constructing requests to managed login and hosted UI services, see [User pool endpoints and managed login reference](#).

Note

Amazon Cognito managed login doesn't support custom authentication with [custom authentication challenge Lambda triggers](#).

Topics

- [Managed login localization](#)
- [Setting up managed login with AWS Amplify](#)
- [Setting up managed login with the Amazon Cognito console](#)
- [Viewing your sign-in page](#)
- [Customizing your authentication pages](#)
- [Things to know about managed login and the hosted UI](#)
- [Configuring a user pool domain](#)
- [Apply branding to managed login pages](#)

Managed login localization

Managed login defaults to the English language in user-interactive pages. You can display your managed login pages localized for the language of your choice. The available languages are those available in the AWS Management Console. In the link that you distribute to users, add a `lang` query parameter, as shown in the following example.

```
https://<your domain>/oauth2/authorize?lang=es&response_type=code&client_id=<your app client id>&redirect_uri=<your relying-party url>
```

Amazon Cognito sets a cookie in users' browser with their language preference after the initial request with a `lang` parameter. After the cookie is set, the language selection persists without displaying or requiring you to include the parameter in requests. For example, after a user makes a sign-in request with a `lang=de` parameter, their managed login pages render in German until they clear their cookies or make a new request with a new localization parameter like `lang=en`.

Localization is only available for managed login. You must be on the Essentials or Plus [feature plan](#) and have assigned your domain to use [managed login branding](#).

The selection that your user makes in managed login isn't available to [custom email or SMS sender triggers](#). When you implement these triggers, you must use other mechanisms, for example the `locale` attribute, to determine a user's preferred language.

The following languages are available.

Managed login languages

Language	Code
German	de
English	en
Spanish	es
French	fr
Bahasa Indonesia	id
Italian	it
Japanese	ja
Korean	ko
Portuguese	pt-BR
Chinese (Simplified)	zh-CN
Chinese (Traditional)	zh-TW

Setting up managed login with AWS Amplify

If you use AWS Amplify to add authentication to your web or mobile app, you can set up your managed login pages in the Amplify command line interface (CLI) and libraries in the Amplify framework. To add authentication to your app, add the Auth category to your project. Then, in your application, authenticate user pool users with Amplify client libraries.

You can invoke managed login pages for authentication or you can federate users through an authorization endpoint that redirects to an IdP. After a user successfully authenticates with the provider, Amplify creates a new user in your user pool and passes the user's tokens to your app.

The following examples show how to use AWS Amplify to set up managed login with social providers in your app.

- [AWS Amplify authentication for JavaScript.](#)

- [AWS Amplify authentication for Swift.](#)
- [AWS Amplify authentication for Flutter.](#)
- [AWS Amplify authentication for Android.](#)

Setting up managed login with the Amazon Cognito console

The first requirement for managed login and hosted UI is a user pool domain. In the user pools console, navigate to the **Domain** tab of your user pool and add a **Cognito domain** or a **custom domain**. You can also choose a domain during the process of creating a new user pool. For more information, see [Configuring a user pool domain](#). When a domain is active in your user pool, all app clients serve public authentication pages on that domain.

When you create or modify a user pool domain, you set the **Branding version** for your domain. This branding version is a choice of **Managed login** or **Hosted UI (classic)**. Your choice of branding version applies to all app clients that use the sign-in services at your domain.

The next step is to create an [app client](#) from the **App clients** tab of your user pool. In the process of creating an app client, Amazon Cognito will ask you for information about your application, then prompt you to select a **Return URL**. The return URL is also called the relying party (RP) URL, the redirect URI, and the callback URL. This is the URL that your application runs from, for example <https://www.example.com> or myapp://example.

After you configure a domain and app client with a branding style in your user pool, your managed login pages become available on the internet.

Viewing your sign-in page

In the Amazon Cognito console, choose the **View login pages** button in the **Login pages** tab for your app client under the **App clients** menu. This button takes you to a sign-in page in your user pool domain with the following basic parameters.

- The app client id
- An authorization code grant request
- A request for all scopes that you have activated for the current app client
- The first callback URL in the list for the current app client

The **View login page** button is useful when you want to test the basic functions of your managed login pages. Your login pages will match the **Branding version** that you assigned to your [user pool domain](#). You can customize your sign-in URL with additional and modified parameters. In most cases, the automatically-generated parameters of the **View login page** link don't fully match the needs of your app. In these cases, you must customize the URL that your app invokes when it signs in your users. For more information about sign-in parameter keys and values, see [User pool endpoints and managed login reference](#).

The sign-in webpage uses the following URL format. This example requests an authorization code grant with the `response_type=code` parameter.

```
https://<your domain>/oauth2/authorize?response_type=code&client_id=<your app client id>&redirect_uri=<your relying-party url>
```

You can look up your user pool domain string from the **Domain** menu in your user pool. In the **App clients** menu, you can identify app client IDs, their callback URLs, their allowed scopes, and other configuration.

When you navigate to the `/oauth2/authorize` endpoint with your custom parameters, Amazon Cognito either redirects you to the `/oauth2/login` endpoint or, if you have an `identity_provider` or `idp_identifier` parameter, silently redirects you to your IdP sign-in page.

Example request for an implicit grant

You can view your 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://mydomain.us-east-1.amazonaws.com/authorize?  
response_type=token&client_id=lexample23456789&redirect_uri=https://  
mydomain.example.com
```

The identity and access tokens appear as parameters appended to your redirect URL.

The following is an example response from an implicit grant request.

`https://mydomain.example.com/
#id_token=eyJraaBcDeF1234567890&access_token=eyJraGhIjKLM1112131415&expires_in=3600&token_type=`

Customizing your authentication pages

In the past, Amazon Cognito only hosted login pages with the classic *hosted UI*, a simple design that grants a universal look to authentication webpages. You could customize Amazon Cognito user pools with a logo image and tweak some styles with a file that specified some CSS style values. Later, Amazon Cognito introduced *managed login*, an updated hosted authentication service. Managed login is an updated look-and-feel with the *branding designer*. The branding designer is a no-code visual editor and a larger suite of options than the hosted UI customization experience. Managed login also introduced custom background images and a dark mode theme.

You can switch between the managed login and hosted UI branding experiences in user pools. To learn more about customizing your managed login pages, see [Apply branding to managed login pages](#).

Things to know about managed login and the hosted UI

The one-hour managed login and hosted UI session cookie

When a user signs in with your login pages or a third-party provider, Amazon Cognito sets a cookie in their browser. With this cookie, users can sign in again with the same authentication method for one hour. When they sign in with their browser cookie, they get fresh tokens that last the duration specified in your app client configuration. Changes to user attributes or authentication factors have no effect on their ability to sign in again with their browser cookie.

Authentication with the session cookie doesn't reset the cookie duration to an additional hour. Users must sign in again if they attempt to access your sign-in pages more than an hour after their most recent successful interactive authentication.

Confirming user accounts and verifying user attributes

For user pool [local users](#), managed login and the hosted UI work best when you configure your user pool to **Allow Cognito to automatically send messages to verify and confirm**. When you enable this setting, Amazon Cognito sends a message with a confirmation code to users who sign up. When you instead confirm users as a user pool administrator, your login pages display an

error message after sign-up. In this state, Amazon Cognito has created the new user, but hasn't been able to send a verification message. You can still confirm users as an administrator, but they might contact your support desk after they encounter an error. For more information about administrative confirmation, see [Allowing users to sign up in your app but confirming them as a user pool administrator](#).

Viewing your changes to configuration

If you make style changes to your pages and they do not immediately appear, wait a few minutes and then refresh the page.

Decoding user pool tokens

Amazon Cognito user pool tokens are signed using an RS256 algorithm. You can decode and verify user pool tokens using AWS Lambda. See [Decode and verify Amazon Cognito JWT tokens](#) on GitHub.

TLS version

Managed login and hosted UI pages require encryption in transit. User pool domains that are provided by Amazon Cognito require that users' browsers negotiate a minimum TLS version of 1.2. Custom domains support browser connections with TLS version 1.2. The hosted UI (classic) **doesn't require** TLS 1.2 for custom domains, but the newer managed login **requires** TLS version 1.2 both for custom and prefix domains. Because Amazon Cognito manages the configuration of your domain services, you can't modify the TLS requirements of your user pool domain.

CORS policies

Neither managed login nor the hosted UI support custom cross-origin resource sharing (CORS) origin policies. A CORS policy would prevent users from passing authentication parameters in their requests. Instead, implement a CORS policy in your application front end. Amazon Cognito returns an `Access-Control-Allow-Origin: *` response header to requests to the following endpoints.

1. [Token endpoint](#)
2. [Revoke endpoint](#)
3. [userInfo endpoint](#)

Cookies

Managed login and the hosted UI set cookies in users' browsers. The cookies conform to the requirements of some browsers that sites not set third-party cookies. They are scoped only to your user pool endpoints and include the following:

- An XSRF-TOKEN cookie for each request.
- A csrf-state cookie for session consistency when a user is redirected.
- A cognito session cookie that preserves successful sign-in attempts for an hour.
- A lang cookie that preserves a user's choice of [language localization](#) in managed login.
- A page-data cookie that persists required data as a user navigates between managed login pages.

In iOS, you can [block all cookies](#). This setting isn't compatible with managed login or the hosted UI. To work with users who might enable this setting, build user pool authentication into a native iOS app with an AWS SDK. In this scenario, you can build your own session storage that isn't cookie-based.

Effects of managed login version change

Consider the following effects of adding domains and setting the managed login version.

- When you add a prefix domain, either with managed login or hosted UI (classic) branding, it can take up to 60 seconds before your login pages are available.
- When you add a custom domain, either with managed login or hosted UI (classic) branding, it can take up to five minutes before your login pages are available.
- When you change the branding version for your domain, it can take up to four minutes before your login pages are available in the new branding version.
- When you switch between managed login and hosted UI (classic) branding, Amazon Cognito doesn't maintain user sessions. They must sign in again with the new interface.

Default style

When you create an app client in the AWS Management Console, Amazon Cognito automatically assigns a branding style to your app client. When you programmatically create an app client with the [CreateUserPoolClient](#) operation, no branding style is created. Managed login isn't available for an app client created with an AWS SDK until you create one with a [CreateManagedLoginBranding](#) request.

Configuring a user pool domain

Configuring a domain is an optional part of setting up a user pool. A user pool domain hosts features for user authentication, federation with third-party providers, and OpenID Connect (OIDC) flows. It has *managed login*, a prebuilt interface for key operations like signing up, signing in, and password recovery. It also hosts the standard OpenID Connect (OIDC) endpoints like [authorize](#), [userInfo](#), and [token](#), for machine-to-machine (M2M) authorization and other OIDC and OAuth 2.0 authentication and authorization flows.

Users authenticate with managed login pages at the domain associated with your user pool. You have two options for configuring this domain: you can either use the default Amazon Cognito hosted domain, or you can configure a custom domain that you own.

The custom domain option has more options for flexibility, security and control. For example, a familiar, organization-owned domain can encourage user trust and make the sign-in process more intuitive. However, the custom domain approach requires some additional overhead, like managing the SSL certificate and DNS configuration.

The OIDC discovery endpoints, `/.well-known/openid-configuration` for endpoint URLs and `/.well-known/jwks.json` for token signing keys, aren't hosted on your domain. For more information, see [Identity provider and relying party endpoints](#).

Understanding how to configure and manage the domain for your user pool is an important step in integrating authentication into your application. Sign-in with the user pools API and an AWS SDK can be an alternative to configuring a domain. The API-based model delivers tokens directly in an API response, but for implementations that use the extended capabilities of user pools as an OIDC IdP, you must configure a domain. For more information about the authentication models that are available in user pools, see [Understanding API, OIDC, and managed login pages authentication](#).

Topics

- [Things to know about user pool domains](#)
- [Using the Amazon Cognito prefix domain for managed login](#)
- [Using your own domain for managed login](#)

Things to know about user pool domains

User pool domains are a point of service for OIDC relying parties in your applications and for UI elements. Consider the following details when you're planning your implementation of a domain for your user pool.

Reserved terms

You can't use the text aws, amazon, or cognito in the name of an Amazon Cognito prefix domain.

Discovery endpoints are on a different domain

The user pool [discovery endpoints](#) .well-known/openid-configuration and .well-known/jwks.json aren't on your user pool custom or prefix domain. The path to these endpoints is as follows.

- `https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/openid-configuration`
- `https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/jwks.json`

Effective time of domain changes

It can take Amazon Cognito up to a minute to launch or update the branding version of a prefix domain. Changes to a custom domain can take up to five minutes to propagate. New custom domains can take up to one hour to propagate.

Custom and prefix domains at the same time

You can set up a user pool with both a custom domain and a prefix domain that's owned by AWS. Because the user pool [discovery endpoints](#) are hosted at a different domain, they only serve the *custom domain*. For example, your openid-configuration will provide a single value for "authorization_endpoint" of "https://auth.example.com/oauth2/authorize".

When you have both custom and prefix domains in a user pool, you can use the custom domain with the full features of an OIDC provider. The prefix domain in a user pool with this configuration doesn't have discovery or token-signing-key endpoints and should be used accordingly.

Custom domains preferred as relying party ID for passkey

When you set up user pool authentication with [passkeys](#), you must set a relying party (RP) ID. When you have a custom domain and a prefix domain, you can set the RP ID only as your custom domain. To set a prefix domain as the RP ID in the Amazon Cognito console, delete your custom domain or enter the fully-qualified domain name (FQDN) of the prefix domain as a **Third-party domain**.

Don't use custom domains at different levels of your domain hierarchy

You can configure separate user pools to have custom domains in the same top-level domain (TLD), for example *auth.example.com* and *auth2.example.com*. The managed login session cookie is valid for a custom domain and all subdomains, for example **.auth.example.com*. Because of this, no user of your applications should access managed login for any parent domain *and* subdomain. Where custom domains use the same TLD, keep them at the same subdomain level.

Say you have a user pool with the custom domain *auth.example.com*. Then you create another user pool and assign the custom domain *uk.auth.example.com*.. A user signs in with *auth.example.com*. and gets a cookie that their browser presents to any website in the wildcard path **.auth.example.com*. They then try to sign in to *uk.auth.example.com*.. They pass an invalid cookie to your user pool domain and receive an error instead of a sign-in prompt. By contrast, a user with a cookie for **.auth.example.com* has no issues starting a sign-in session at *auth2.example.com*.

Branding version

When you create a domain, you set a **Branding version**. Your options are the newer managed-login experience and the classic hosted UI experience. This choice applies to all app clients that host services at your domain.

Using the Amazon Cognito prefix domain for managed login

The default experience for managed login is hosted on a domain that AWS owns. This approach has a low barrier to entry—choose a prefix name and it's active—but doesn't have the trust-inspiring features of a custom domain. There isn't a cost difference between the Amazon Cognito domain option and the custom domain option. The only difference is the domain in the web address that you direct your users to. For cases of third-party IdP redirects and client-credentials flows, the hosted domain has little visible effect. A custom domain is better for cases where your users sign in with managed login and would interact with a authentication domain that doesn't match the application domain.

The hosted Amazon Cognito domain has a prefix of your choosing, but is hosted at the root domain `amazoncognito.com`. The following is an example:

`https://cognitoexample.auth.ap-south-1.amazoncognito.com`

All prefix domains follow this format: `prefix.auth.AWS Region code.amazoncognito.com`. [Custom domain](#) user pools can host the managed login or hosted UI pages on any domain that you own.

Note

To augment the security of your Amazon Cognito applications, the parent domains of user pool endpoints are registered in the [Public Suffix List \(PSL\)](#). The PSL helps your users' web browsers establish a consistent understanding of your user pool endpoints and the cookies they set.

User pool parent domains take the following formats.

`auth.Region.amazoncognito.com`
`auth-fips.Region.amazoncognito.com`

To add an app client and a user pool domain with the AWS Management Console, see [Creating an app client](#).

Topics

- [Prerequisites](#)
- [Configure an Amazon Cognito domain prefix](#)
- [Verify your sign-in page](#)

Prerequisites

Before you begin, you need:

- A user pool with an app client. For more information, see [Getting started with user pools](#).

Configure an Amazon Cognito domain prefix

You can use either the AWS Management Console or the AWS CLI or API to configure a user pool domain.

Amazon Cognito console

Configure a domain

1. Navigate to the **Domain** menu under **Branding**.
2. Next to **Domain**, choose **Actions** and select **Create Cognito domain**. If you have already configured a user pool prefix domain, choose **Delete Cognito domain** before creating your new custom domain.
3. Enter an available domain prefix to use with a **Amazon Cognito domain**. For information on setting up a **Custom domain**, see [Using your own domain for managed login](#).
4. Choose a **Branding version**. Your branding version applies to all user-interactive pages at that domain. Your user pool can host either managed login or hosted UI branding for all app clients.

 **Note**

You can have a custom domain and a prefix domain, but Amazon Cognito only serves the `/well-known/openid-configuration` endpoint for the *custom* domain.

5. Choose **Create**.

CLI/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-identity create-user-pool-domain`

Example: `aws cognito-identity create-user-pool-domain --user-pool-id <user_pool_id> --domain <domain_name> --managed-login-version 2`

- User pools API operation: [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>

- User pools API operation: [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>

- User pools API operation: [DeleteUserPoolDomain](#)

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 managed login

After you set up an app client, you can configure your user pool with a custom domain for the domain services of [managed login](#). With a custom domain, users can sign in to your application using your own web address instead the default [amazoncognito.com prefix domain](#). Custom domains improve user trust in your application with a familiar domain name, especially when the root domain matches the domain that hosts your application. Custom domains can improve compliance with organizational security requirements.

A custom domain has some prerequisites, including a user pool, an app client, and a web domain that you own. Custom domains also require an SSL certificate for the custom domain, managed with AWS Certificate Manager (ACM) in US East (N. Virginia). Amazon Cognito creates a Amazon

CloudFront distribution, secured in transit with your ACM certificate. Because you own the domain, you must create a DNS record that directs traffic to the CloudFront distribution for your custom domain.

After these elements are ready, you can add the custom domain to your user pool through the Amazon Cognito console or API. This involves specifying the domain name and SSL certificate, and then updating your DNS configuration with the provided alias target. After making these changes, you can verify that the sign-in page is accessible at your custom domain.

The lowest-effort way to create a custom domain is with a public hosted zone in Amazon Route 53. The Amazon Cognito console can create the right DNS records in a few steps. Before you begin, consider [creating a Route 53 hosted zone](#) for a domain or subdomain that you own.

Topics

- [Adding a custom domain to a user pool](#)
- [Prerequisites](#)
- [Step 1: Enter your custom domain name](#)
- [Step 2: Add an alias target and subdomain](#)
- [Step 3: Verify your sign-in page](#)
- [Changing the SSL certificate for your custom domain](#)

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](#).
- A web domain that you own. Its *parent domain* must have a valid DNS **A record**. You can assign any value to this record. The parent may be the root of the domain, or a child domain that is one step up in the domain hierarchy. For example, if your custom domain is *auth.xyz.example.com*, Amazon Cognito must be able to resolve *xyz.example.com* to an IP address. To prevent accidental

impact on customer infrastructure, Amazon Cognito doesn't support the use of top-level domains (TLDs) for custom domains. For more information see [Domain Names](#).

- The ability to create a subdomain for your custom domain. We recommend **auth** for your subdomain name. 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 public SSL/TLS certificate managed by ACM in US East (N. Virginia). The certificate must be in us-east-1 because the certificate will be associated with a distribution in CloudFront, a global service.
- An application that permits your user pool authorization server to add cookies to user sessions. Amazon Cognito sets several required cookies for managed login pages. These include cognito, cognito-f1, and XSRF-TOKEN. Although each individual cookie conforms to browser size limits, changes to your user pool configuration might cause managed login cookies to grow in size. An intermediate service like an Application Load Balancer (ALB) in front of your custom domain might enforce a maximum header size or total cookie size. If your application also sets its own cookies, your users' sessions might exceed these limits. We recommend that, to avoid size limit conflicts, your application not set cookies on the subdomain that hosts your user pool domain services.
- Permission to update Amazon CloudFront distributions. You can do so by attaching the following IAM policy statement to a user in your AWS account:

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

{}

For more information about authorizing actions in CloudFront, see [Using Identity-Based Policies \(IAM Policies\) for CloudFront](#).

Amazon Cognito initially uses your IAM permissions to configure the CloudFront distribution, but the distribution is managed by AWS. You can't change the configuration of the CloudFront distribution that Amazon Cognito associated with your user pool. For example, you can't update the supported TLS versions in the security policy.

Step 1: Enter your custom domain name

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

Amazon Cognito console

To add your domain to your user pool from the Amazon Cognito console:

1. Navigate to the **Domain** menu under **Branding**.
2. Next to **Domain**, choose **Actions** and select **Create custom domain** or **Create Amazon Cognito domain**. If you have already configured a user pool custom domain, choose **Delete custom domain** before creating your new custom domain.
3. Next to **Domain**, choose **Actions** and select **Create custom domain**. If you have already configured a custom domain, choose **Delete custom domain** to delete the existing domain before creating your new custom domain.
4. For **Custom domain**, enter the URL of the domain you want to use with Amazon Cognito. 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.
5. For **ACM certificate**, choose the SSL certificate that you want to use for your domain. Only ACM certificates in US East (N. Virginia) are eligible to use with an Amazon Cognito custom domain, regardless of the AWS Region of your user pool.

If you don't have an available certificate, you can use ACM to provision one in US East (N. Virginia). For more information, see [Getting Started](#) in the *AWS Certificate Manager User Guide*.

6. Choose a **Branding version**. Your branding version applies to all user-interactive pages at that domain. Your user pool can host either managed login or hosted UI branding for all app clients.

 **Note**

You can have a custom domain and a prefix domain, but Amazon Cognito only serves the `/.well-known/openid-configuration` endpoint for the *custom* domain.

7. Choose **Create**.
8. Amazon Cognito returns you to the **Social and external providers** menu. A message titled **Create an alias record in your domain's DNS** is displayed. Note down the **Domain** and **Alias target** displayed in the console. They will be used in the next step to direct traffic to your custom domain.

API

The following [CreateUserPoolDomain](#) request body creates a custom domain.

```
{  
    "Domain": "auth.example.com",  
    "UserPoolId": "us-east-1_EXAMPLE",  
    "ManagedLoginVersion": 2,  
    "CustomDomainConfig": {  
        "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/  
a1b2c3d4-5678-90ab-cdef-EXAMPLE1111"  
    }  
}
```

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 must use your DNS service provider's configuration tools to add the alias target from the previous step to your domain's DNS record. 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](#). If prompted, enter your AWS credentials.
2. If you don't have a public hosted zone in Route 53, create one with a root that is a parent of your custom domain. For more information, see [Creating a public hosted zone](#) in the *Amazon Route 53 Developer Guide*.
 - a. Choose **Create Hosted Zone**.
 - b. Enter the parent domain, for example *auth.example.com*, of your custom domain, for example *myapp.auth.example.com*, from the **Domain Name** list.
 - c. Enter a **Description** for your hosted zone.
 - d. Choose a hosted zone **Type** of **Public hosted zone** to allow public clients to resolve your custom domain. Choosing **Private hosted zone** is not supported.
 - e. Apply **Tags** as desired.
 - f. Choose **Create hosted zone**.

 **Note**

You can also create a new hosted zone for your custom domain with a delegation set in the parent hosted zone that directs queries to the subdomain hosted zone. Otherwise, create an A record. This method offers more flexibility and security with your hosted zones. For more information, see [Creating a subdomain for a domain hosted through Amazon Route 53](#).

3. On the **Hosted Zones** page, choose the name of your hosted zone.
4. Add a DNS record for the parent domain of your custom domain, if you don't already have one. Create a DNS record for the parent domain with the following properties:
 - **Record name:** Leave blank.

- **Record type:** A.
 - **Alias:** Don't enable.
 - **Value:** Enter a target of your choosing. This record must resolve to *something*, but the value of the record doesn't matter to Amazon Cognito.
 - **TTL:** Set to your preferred TTL or leave as default.
 - **Routing policy:** Choose **Simple routing**.
5. Choose **Create records**. The following is an example record for the domain *example.com*:

example.com. 60 IN A 198.51.100.1

 **Note**

Amazon Cognito verifies that there is a DNS record for the parent domain of your custom domain to protect against accidental hijacking of production domains. If you do not have a DNS record for the parent domain, Amazon Cognito will return an error when you attempt to set the custom domain. A Start of Authority (SOA) record isn't a sufficient DNS record for the purposes of parent-domain verification.

6. Add another DNS record for your custom domain with the following properties:
- **Record name:** Your custom domain prefix, for example auth to create a record for auth.example.com.
 - **Record type:** A.
 - **Alias:** Enable.
 - **Route traffic to:** Choose **Alias to Cloudfront distribution**. Enter the **Alias target** you recorded earlier, for example 123example.cloudfront.net.
 - **Routing policy:** Choose **Simple routing**.
7. Choose **Create records**.

 **Note**

Your new records can take around 60 seconds to propagate to all Route 53 DNS servers. You can use the Route 53 [GetChange](#) API method to verify that your changes have propagated.

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://myapp.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.

AWS Management Console

To renew a certificate from the 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 **User Pools**.
3. Choose the user pool for which you want to update the certificate.
4. Choose the **Social and external providers** menu.
5. Choose **Actions, Edit ACM certificate**.
6. Select the new certificate you want to associate with your custom domain.
7. Choose **Save changes**.

API

To renew a certificate (Amazon Cognito API)

- Use the [UpdateUserPoolDomain](#) action.

Apply branding to managed login pages

You might want to provide a consistent user experience between your authentication service and your application. You can accomplish this goal either with custom forms and back-end API operations in an AWS SDK, or with managed login. Managed login and the classic hosted UI are web front ends for the component of your application that serves authentication with user pools. To synchronize your managed authentication services with your application UX, you have two customization options: the branding designer and hosted UI branding. You can choose your preferred experience in the Amazon Cognito console and with user pool API operations.

The branding designer

The [branding designer](#) is the newest customization option for the newest user pools UI experience, [managed login](#). The branding designer is a no-code visual editor for managed login assets and style, and a set of API operations for programmatic configuration of a large number of configuration options. User pools that you configure with a [domain](#) and managed login automatically render the branding-designer version of your login pages.

Hosted UI (classic) branding

The [hosted UI \(classic\) branding experience](#) has two options: to modify a cascading stylesheets (CSS) file with a fixed set of style options, and to add a custom logo image. You can set these options in the Amazon Cognito console or with the [SetUICustomization](#) API operation. At the time that the service launched, Amazon Cognito had only this option. User pools that you configure with a [domain](#) and the hosted UI branding version automatically render the classic version of your login pages. Your [feature plan](#) might also support only the hosted UI.

Choose a branding experience and assign styles

In the Amazon Cognito console, new user pools default to the **Managed login** branding experience. User pools that you set up before managed login was available will have **Hosted UI (classic)** branding. You can switch between managed login and hosted UI branding. When you change your **Branding version**, Amazon Cognito immediately applies the change to the user-interactive pages of your user pool domain. With managed login and the hosted UI, your user pool can have a style for each app client.

Each app client can have a distinct branding *style*, but a user pool domain serves either managed login or the hosted UI. A style is the set of customization settings applied to an app client. You can set up one [custom domain](#) and one [prefix domain](#) per user pool. You can assign different branding versions to your custom and prefix domains. However, a prefix domain isn't fully functional when you also have a custom domain—the .well-known OIDC discovery endpoints *only* present custom-domain paths. You can only use the prefix domain for operations that don't require endpoint discovery (`openid-configuration`) in a user pool with this configuration. Because of these properties of user pools, you can effectively choose one branding version per user pool.

You can assign styles to the app clients in a user pool where a domain is set to the managed login branding version. Styles are a set of visual settings made up of image files, display options, CSS values. When you assign a style to an app client, Amazon Cognito immediately pushes your updates to your user-interactive login pages. Amazon Cognito renders your user-interactive pages with your chosen branding version and the customization that you have applied to it.

Update and delete styles

When you create a style, you link it to an app client. To change a style assignment for an app client, you must first delete the original style. Currently, you can't copy settings between styles. You must do this programmatically. To replicate settings between styles and app clients, get the

settings for a style with the [DescribeManagedLoginBranding](#) API operation and apply them with [CreateManagedLoginBranding](#) or [UpdateManagedLoginBranding](#). You can't change the assigned styles of an app client—you can only delete the original and set a new one. For more information about managing styles with API and SDK operations, see [API and SDK operations for managed login branding](#).

Note

Programmatic requests that create or update branding style must have a request size of no more than 2 MB. If your request is larger than this limit, break your request up into multiple `UpdateManagedLoginBranding` requests for groups of parameters that don't exceed the maximum request size. These requests don't result in unspecified parameters being set to default, so you can send partial requests without any effect on existing settings.

You delete a style in the Amazon Cognito console from the **Managed login** menu. Under **Styles**, choose the style that you want to delete and choose **Delete style**.

At a high level, the process of assigning branding to a domain consists of the following steps.

1. [Create a domain and set the branding version](#).
2. Create a branding style and assign it to an app client.

To assign a style to an app client

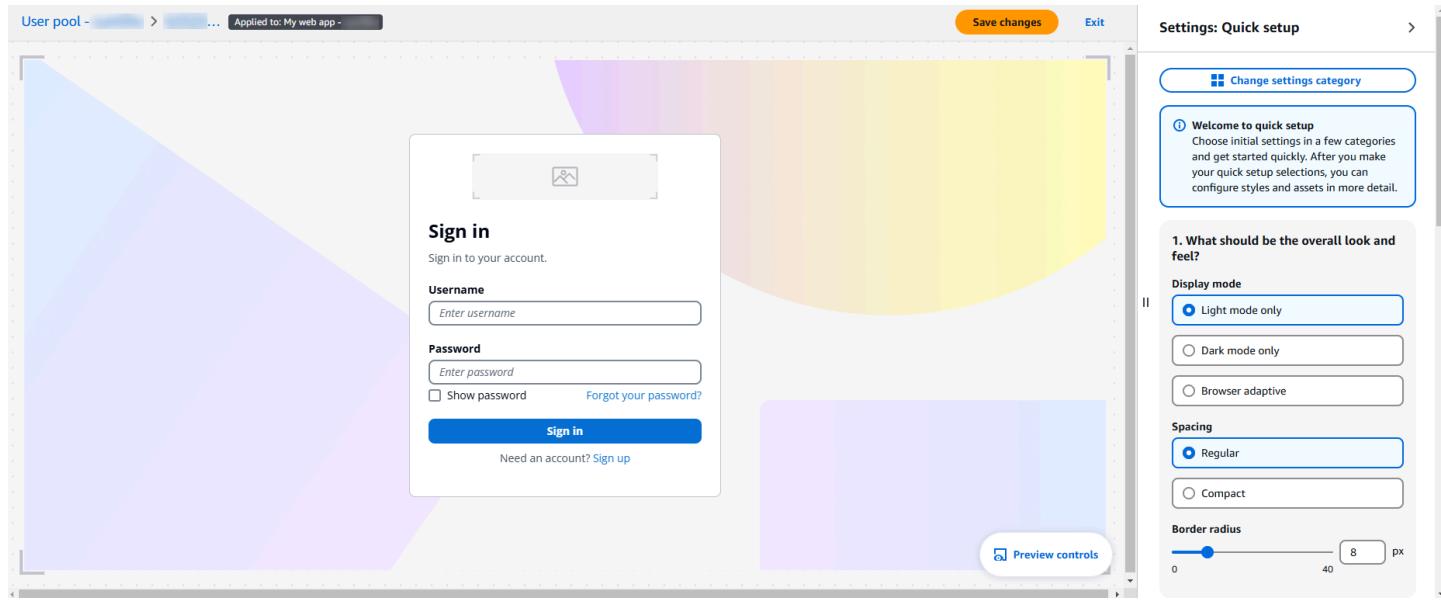
1. In the **Domain** menu of your user pool, create a domain and set the **Branding version** to **Managed login**.
2. Navigate to the **Managed login** menu. Under **Styles**, choose **Create a style**.
3. Choose the app client that you want to assign your style to, or create a new [app client](#).
4. To start configuring your branding settings, choose **Launch branding designer**.

Topics

- [The branding designer and customizing managed login](#)
- [Customizing hosted UI \(classic\) branding](#)

The branding designer and customizing managed login

The branding designer is a visual design and editing tool for your managed login webpages. It's built in to the Amazon Cognito console. In the branding designer, you start with a preview of your login pages and can proceed into a quick-setup option or a detailed view with advanced options. You can modify and preview style parameters or add a custom background image and logo. You can configure light mode and dark mode.



To begin, create a style that you can apply to your user pool or an app client.

To get started with the branding designer

1. [Create a domain](#) from the **Domain** tab, or update your existing domain. Under **Branding version**, set your domain to use **Managed login**.
2. Delete the existing app client style, if any.
 - a. In the **App clients** menu, select your app client.
 - b. Under **Managed login style**, select the style assigned to your app client.
 - c. Choose **Delete style**. Confirm your selection.
3. Navigate to the **Managed login** menu in your user pool. If you haven't already, follow the prompt to select a [feature plan](#) that includes managed login. You can also select **Preview this feature** if you want to check out the branding designer without making changes.
4. Under **Styles**, choose **Create a style**.

5. Choose the app client that you want to assign your style to and select **Create**. You can also create a new app client.
6. The Amazon Cognito console launches the branding designer.
7. Choose a tab where you want to start editing, or select **Launch editor** and enter [quick setup](#). The following tabs are available:

Preview

See how your current selections look in your managed login pages.

Foundation

Set an overall theme, configure links to external identity providers, and style form fields.

Components

Configure styles for headers, footers, and individual UI elements.

8. To make choices about initial settings, enter quick setup. Select **Change settings category** and choose **Quick setup**. When you select **Proceed**, the branding designer launches with a set of basic options for you to configure.

Quick setup

The **Launch branding designer** button loads a visual editor for your managed login configuration where you can select from a variety of primary customization options. As you make selections, Amazon Cognito renders your managed login changes in a preview window. To return to the detailed settings menu, select the **Change settings category** button.

What should be the overall look and feel?

Configure basic theme settings for managed login.

Display mode

Choose a light-mode, dark-mode, or adaptive experience for your managed login. The adaptive settings defers to the user's browser preference when Amazon Cognito renders managed login. When you choose a browser-adaptive mode, you can choose different colors and logo images for light and dark mode.

Spacing

Set the default spacing between elements in the page.

Border radius

Set the rounding depth of the outer border of elements.

How should the page background look?

Background type

The **Show image** checkbox indicates whether you want a background image or to set a solid background color.

1. To use an image, select **Show image** and choose a background image for light and dark modes. You can also set a dark-mode and light-mode **Page background color** for areas of the background that aren't covered by the image.
2. To use only a color for the background, deselect **Show image** and choose a light-mode and dark-mode **Page background color**.

How should forms look?

Configure settings for the form elements of managed login. Examples of form items are login and code prompts.

Horizontal alignment

Set the horizontal alignment of form fields.

Form logo

Set the positioning of your logo image.

Logo image

Choose a logo image file to include in the form element for light and dark modes. To upload an image, select the **Logo image** dropdown, choose **Add new asset**, and add a logo file.

Primary branding color

Set a theme color for light and dark modes. This color will be applied as the background color to all elements classified as primary.

How should headers look?

Choose whether you want to include a header in your managed login pages. The header can contain a logo image.

Header logo

Set the position of the logo image in your header.

Logo image

Choose a logo position and a logo image file to include in the header. To upload an image, select the **Logo image** dropdown, choose **Add new asset**, and add a logo file.

Header background color

Set the light and dark mode colors for the background of the header.

Detailed settings

In the detailed settings view, you can modify individual components in the **Foundation** and **Components**. The **Preview** tab displays a preview of managed login in the current context with your customizations.

Amazon Cognito > User pools > User pool - [REDACTED] > Managed login > Style:

Style: [REDACTED] Info Delete style Launch branding designer

General information Info

Assigned app client
My web app - [REDACTED]

Branding customizations
Cognito default settings

Last customized time
November 11, 2024 at 11:19 PST

Preview Foundation Components

The screenshot shows the 'Preview' tab selected in the theme studio editor. The preview window displays a 'Sign in' page with a light gray header containing a camera icon, a main title 'Sign in', and a subtitle 'Sign in to your account.'. Below the title are two input fields: 'Username' with placeholder 'Enter username' and 'Password' with placeholder 'Enter password'. To the right of the password field is a checkbox labeled 'Show password' and a link 'Forgot your password?'. A large blue button at the bottom is labeled 'Sign in'. At the bottom of the page, there is a link 'Need an account? Sign up'. The background of the preview window features abstract, semi-transparent geometric shapes in purple, yellow, and pink.

To enter the visual editor for a component, choose the edit icon in the tile for the component. From the theme studio editor, you can switch between components with the **Change setting category** button.

Foundation

App style

Configure the basics of your managed login configuration. This category has settings for the overall theme, text spacing, and the page header and footer.

Display mode

Choose a light-mode, dark-mode, or adaptive experience for your managed login pages. When you choose a browser-adaptive mode, you can choose different colors and logo images for light and dark mode.

Spacing

Set the default spacing between elements in the page.

Authentication behavior

Configure styles for the buttons that connect your users to external identity providers (IdPs).

Form behavior

Configure styles for input forms: the positioning of inputs, colors, and alignment of elements.

Components

Buttons

Styles for buttons that Amazon Cognito renders on managed login pages.

Divider

Styles for divider lines and boundaries between managed login elements like the input form and the external-provider sign-in selector.

Dropdown

Styles for dropdown menus.

Favicon

Styles for the image that Amazon Cognito provides for the tab and bookmark icon.

Focus rings

Styles for the highlights that indicate a currently-selected input.

Form container

Styles for the elements that bound a form.

Global footer

Styles for the footer that Amazon Cognito displays at the bottom of managed login pages.

Global header

Styles for the header that Amazon Cognito displays at the top of managed login pages.

Indications

Styles for error and success messages.

Option controls

Styles for checkboxes, multi-selects, and other input prompts.

Page background

Styles for the overall background of managed login.

Inputs

Styles for form-field input prompts.

Link

Styles for hyperlinks in managed login pages.

Text for page

Styles for in-page text.

Text for field

Styles for the text around form inputs.

API and SDK operations for managed login branding

You can also apply branding to a managed login style with the API operations

[CreateManagedLoginBranding](#) and [UpdateManagedLoginBranding](#). These operations are ideal for creating identical or slightly-modified versions of a branding style for another app client or user pool. Query the managed login branding of an existing style with the API operation [DescribeManagedLoginBranding](#), then modify the output as needed and apply it to another resource.

The `UpdateManagedLoginBranding` operation doesn't change the app client that your style is applied to. It only updates the existing style that's assigned to an app client. To completely replace the style for an app client, delete the existing style with [DeleteManagedLoginBranding](#) and assign a new style with [CreateManagedLoginBranding](#). In the Amazon Cognito console, the same is true: you must delete the existing style and create a new one.

Setting up managed login branding in an API or SDK request requires that your settings be embedded in a JSON file that's converted to a Document datatype. The following is guidance for images that you can add and for generating programmatic requests to configure a branding style.

Image assets

[CreateManagedLoginBranding](#) and [UpdateManagedLoginBranding](#) include an `Assets` parameter. This parameter is an array of image files in base64-encoded binary format.

Note

Programmatic requests that create or update branding style must have a request size of no more than 2 MB. The assets in your request might make it exceed this limit. If this is the case, break your request up into multiple `UpdateManagedLoginBranding` requests for groups of parameters that don't exceed the maximum request size. These requests don't result in unspecified parameters being set to default, so you can send partial requests without any effect on existing settings.

Some assets have limitations on the filetypes that you can submit.

Asset	Accepted file extensions
FAVICON_ICO	ico
FAVICON_SVG	svg
EMAIL_GRAPHIC	png, svg, jpeg
SMS_GRAPHIC	png, svg, jpeg
AUTH_APP_GRAPHIC	png, svg, jpeg
PASSWORD_GRAPHIC	png, svg, jpeg

Asset	Accepted file extensions
PASSKEY_GRAPHIC	png, svg, jpeg
PAGE_HEADER_LOGO	png, svg, jpeg
PAGE_HEADER_BACKGROUND	png, svg, jpeg
PAGE_FOOTER_LOGO	png, svg, jpeg
PAGE_FOOTER_BACKGROUND	png, svg, jpeg
PAGE_BACKGROUND	png, svg, jpeg
FORM_BACKGROUND	png, svg, jpeg
FORM_LOGO	png, svg, jpeg
IDP_BUTTON_ICON	ico, svg

Files of the SVG type support the following attributes and elements.

Attributes

```
accent-height, accumulate, additive, alignment-baseline, ascent, attributename,
attributetype, azimuth, basefrequency, baseline-shift, begin, bias, by, class,
clip, clip-path, clip-rule, color, color-interpolation, color-interpolation-
filters, color-profile, color-rendering, cx, cy, d, dx, dy, diffuseconstant,
direction, display, divisor, dur, edgemode, elevation, end, fill, fill-opacity,
fill-rule, filter, filterunits, flood-color, flood-opacity, font-family, font-
size, font-size-adjust, font-stretch, font-style, font-variant, font-weight, fx,
fy, g1, g2, glyph-name, glyphref, gradientunits, gradienttransform, height, href,
id, image-rendering, in, in2, k, k1, k2, k3, k4, kerning, keypoints, keysplines,
keytimes, lang, lengthadjust, letter-spacing, kernelmatrix, kernelunitlength,
lighting-color, local, marker-end, marker-mid, marker-start, markerheight,
markerunits, markerwidth, maskcontentunits, maskunits, max, mask, media,
method, mode, min, name, numoctaves, offset, operator, opacity, order, orient,
orientation, origin, overflow, paint-order, path, pathlength, patterncontentunits,
patternttransform, patternunits, points, preservealpha, preserveaspectratio, r,
rx, ry, radius, refx, refy, repeatcount, repeatdur, restart, result, rotate,
scale, seed, shape-rendering, specularconstant, specularexponent, spreadmethod,
stddeviation, stitchtiles, stop-color, stop-opacity, stroke-dasharray, stroke-
```

```
dashoffset, stroke-linecap, stroke-linejoin, stroke-miterlimit, stroke-opacity, stroke, stroke-width, style, surfacescale, tabindex, targetx, targety, transform, text-anchor, text-decoration, text-rendering, textlength, type, u1, u2, unicode, values, viewbox, visibility, vert-adv-y, vert-origin-x, vert-origin-y, width, word-spacing, wrap, writing-mode, xchannelselector, ychannelselector, x, x1, x2, xmlns, y, y1, y2, z, zoomandpan
```

Elements

```
svg, a, altglyph, altglyphdef, altglyphitem, animatecolor, animatemotion, animatetransform, audio, canvas, circle, clippath, defs, desc, ellipse, filter, font, g, glyph, glyphref, hkern, image, line, lineargradient, marker, mask, metadata, mpath, path, pattern, polygon, polyline, radialgradient, rect, stop, style, switch, symbol, text, textpath, title, tref, tspan, video, view, vkern, feBlend, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feDistantLight, feFlood, feFuncA, feFuncB, feFuncG, feFuncR, feGaussianBlur, feMerge, feMergeNode, feMorphology, feOffset, fePointLight, feSpecularLighting, feSpotLight, feTile, feTurbulence
```

Tools for managed login branding operations

Amazon Cognito manages a file in the [JSON-Schema format](#) for the managed-login branding settings object. The following is how to programmatically update your branding style.

To update branding in the user pools API

1. In the Amazon Cognito console, create a default managed login branding style from the **Managed login** menu of your user pool. Assign it to an app client.
2. Record the ID of the app client that you created the style for, for example 1example23456789.
3. Retrieve the settings for the branding style with a [DescribeManagedLoginBrandingByClient](#) API request with `ReturnMergedResources` set to `true`. The following is an example request body.

```
{  
  "ClientId": "1example23456789",  
  "ReturnMergedResources": true,  
  "UserPoolId": "us-east-1_EXAMPLE"  
}
```

4. Modify the output of `DescribeManagedLoginBrandingByClient` with your customizations.
 - a. The response body is wrapped in a `ManagedLoginBranding` element that isn't part of the syntax for create and update operations. Remove this top level of the JSON object.
 - b. To replace images, replace the `Bytes` value with the Base64-encoded binary data of each image file.
 - c. To update settings, modify the output of the `Settings` object and include it in your next request. Amazon Cognito ignores any values in your `Settings` object that aren't in the schema that you receive in your API response.
5. Use the updated response body in a [CreateManagedLoginBranding](#) or [UpdateManagedLoginBranding](#) request. If this request exceeds 2 MB in size, separate it out into multiple requests. These operations work in a PATCH model where original settings remain unchanged unless you specify otherwise.

Customizing hosted UI (classic) branding

You can use the AWS Management Console, or the AWS CLI or API, to specify classic customization settings for the hosted UI. You can upload a custom logo image to be displayed in the app. You can also apply some cascading style sheets (CSS) options to the look and feel of the UI.

You can customize the UI defaults and override individual [app clients](#) with specific settings. Amazon Cognito applies the default configuration to every app client that doesn't have client-level settings.

In the Amazon Cognito console and in API requests, the request that sets your UI customization must not exceed 135 KB in size. In rare cases, the sum of request headers, your CSS file, and your logo might exceed 135KB. Amazon Cognito encodes the image file to Base64. This increases the size of a 100 KB image to 130 KB, keeping five KB for request headers and your CSS. If the request is too large, the AWS Management Console or your SetUICustomization API request returns a `request parameters too large` error. Adjust your logo image to be no greater than 100KB and your CSS file to be no larger than 3 KB. You can't set CSS and logo customization separately.

 **Note**

To customize your UI, you must set up a domain for your user pool.

Specifying a custom logo in classic branding

Amazon Cognito centers your custom logo above the input fields at the [Login endpoint](#).

Choose a PNG, JPG, or JPEG file that can scale to 350 by 178 pixels for your custom hosted UI logo. Your logo file can be no larger than 100 KB in size, or 130 KB after Amazon Cognito encodes to Base64. To set an `ImageFile` in [SetUICustomization](#) in the API, convert your file to a Base64-encoded text string or, in the AWS CLI, provide a file path and let Amazon Cognito encode it for you.

Specifying CSS customizations in classic branding

You can customize the CSS for the hosted app pages, with the following restrictions:

- You can use any of the following CSS class names:
 - `background-customizable`
 - `banner-customizable`
 - `errorMessage-customizable`
 - `idpButton-customizable`
 - `idpButton-customizable:hover`
 - `idpDescription-customizable`
 - `inputField-customizable`
 - `inputField-customizable:focus`
 - `label-customizable`
 - `legalText-customizable`
 - `logo-customizable`
 - `passwordCheck-valid-customizable`
 - `passwordCheck-notValid-customizable`
 - `redirect-customizable`
 - `socialButton-customizable`
 - `submitButton-customizable`
 - `submitButton-customizable:hover`
 - `textDescription-customizable`
- Property values can contain HTML, except for the following values: `@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 of the containing block as a percentage.
- **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 CSS 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 four 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 four 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 four 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 three 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 CSS color value.
- **border-color** is the color of the button border. It can be any standard CSS 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.

Note

When you customize **Legal text**, you are customizing the message **We won't post to any of your accounts without asking first** that is displayed under social identity providers in the sign-in page.

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.

Customizing the hosted UI with classic branding in the 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 **App integration** tab under **Domain**. Your app client ID and callback URL are shown under **App clients**.

To specify app UI customization settings

1. Sign in to the [Amazon Cognito console](#).
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. [Create a domain](#) from the **Domain** tab, or update your existing domain. Under **Branding version**, set your domain to use **Hosted UI (classic)**.
4. Choose the **Managed login** menu.
5. To customize UI settings for all app clients, locate **Style** under **Hosted UI settings** and select **Edit**.
6. To customize UI settings for one app client, go to the **App clients** menu and select the app client you want to modify, then locate **Hosted UI (classic) style** and select **Override**. Select **Edit**.
7. To upload your own logo image file, choose **Choose file** or **Replace current file**.
8. To customize hosted UI CSS, download **CSS template.css** and modify the template with the values you want to customize. Only the keys that are included in the template can be used

with the hosted UI. Added CSS keys will not be reflected in your UI. After you have customized the CSS file, choose **Choose file** or **Replace current file** to upload your custom CSS file.

Customizing the hosted UI with classic branding in the user pools API and with the AWS CLI

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, use the following API operations.

- 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, use the following API operations.

- AWS CLI from image file: `aws cognito-idp set-ui-customization --user-pool-id <your-user-pool-id> --client-id <your-app-client-id> --image-file fileb://"<path-to-logo-image-file>" --css ".label-customizable{ color: <color>;}"`
- AWS CLI with image encoded as Base64 binary text: `aws cognito-idp set-ui-customization --user-pool-id <your-user-pool-id> --client-id <your-app-client-id> --image-file <base64-encoded-image-file> --css ".label-customizable{ color: <color>;}"`
- AWS API: [SetUICustomization](#)

Customizing user pool workflows with Lambda triggers

Amazon Cognito works with AWS Lambda functions to modify the authentication behavior of your user pool. You can configure your user pool to automatically invoke Lambda functions before their first sign-up, after they complete authentication, and at several stages in between. Your functions can modify the default behavior of your authentication flow, make API requests to modify your user pool or other AWS resources, and communicate with external systems. The code in your Lambda functions is your own. Amazon Cognito sends event data to your function, waits for the function to process the data, and in most cases anticipates a response event that reflects any changes you want to make to the session.

Within the system of request and response events, you can introduce your own authentication challenges, migrate users between your user pool and another identity store, customize messages, and modify JSON web tokens (JWTs).

Lambda triggers can customize the response that Amazon Cognito delivers to your user after they initiate an action in your user pool. For example, you can prevent sign-in by a user who would otherwise succeed. They can also perform runtime operations against your AWS environment, external APIs, databases, or identity stores. The migrate user trigger, for example, can combine external action with a change in Amazon Cognito: you can look up user information in an external directory, then set attributes on a new user based on that external information.

When you have a Lambda trigger assigned to your user pool, Amazon Cognito interrupts its default flow to request information from your function. Amazon Cognito generates a JSON *event* and passes it to your function. The event contains information about your user's request to create a user account, sign in, reset a password, or update an attribute. Your function then has an opportunity to take action, or to send the event back unmodified.

The following table summarizes some of the ways you can use Lambda triggers to customize user pool operations:

User Pool Flow	Operation	Description
Custom Authentication Flow	Define Auth Challenge	Determines the next challenge in a custom auth flow
	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	<u>the section called “Pre authentication Lambda trigger”</u>	Custom validation to accept or deny the sign-in request
	<u>the section called “Post authentication Lambda trigger”</u>	Logs events for custom analytics

User Pool Flow	Operation	Description
	<u>the section called “Pre token generation Lambda trigger”</u>	Augments or suppresses token claims
Sign-Up	<u>the section called “Pre sign-up Lambda trigger”</u>	Performs custom validation that accepts or denies the sign-up request
	<u>the section called “Post confirmation Lambda trigger”</u>	Adds custom welcome messages or event logging for custom analytics
	<u>the section called “Migrate user Lambda trigger”</u>	Migrates a user from an existing user directory to user pools
Messages	<u>the section called “Custom message Lambda trigger”</u>	Performs advanced customization and localization of messages
Token Creation	<u>the section called “Pre token generation Lambda trigger”</u>	Adds or removes attributes in Id tokens
Email and SMS third-party providers	<u>the section called “Custom sender Lambda triggers”</u>	Uses a third-party provider to send SMS and email messages

Topics

- [Important considerations](#)
- [Adding a user pool Lambda trigger](#)
- [User pool Lambda trigger event](#)
- [User pool Lambda trigger common parameters](#)
- [Connecting API operations to Lambda triggers](#)
- [Connecting Lambda triggers to user pool functional operations](#)
- [Pre sign-up Lambda trigger](#)

- [Post confirmation Lambda trigger](#)
- [Pre authentication Lambda trigger](#)
- [Post authentication Lambda trigger](#)
- [Custom authentication challenge Lambda triggers](#)
- [Pre token generation Lambda trigger](#)
- [Migrate user Lambda trigger](#)
- [Custom message Lambda trigger](#)
- [Custom sender Lambda triggers](#)

Important considerations

When you are preparing your user pools for Lambda functions, consider the following:

- The events that Amazon Cognito sends to your Lambda triggers might change with new features. The positions of response and request elements in the JSON hierarchy might change, or element names might be added. In your Lambda function, you can expect to receive the input-element key-value pairs described in this guide, but stricter input validation can cause your functions to fail.
- You can choose one of multiple versions of the events that Amazon Cognito sends to some triggers. Some versions might require you to accept a change to your Amazon Cognito pricing. For more information about pricing, see [Amazon Cognito Pricing](#). To customize access tokens in a [Pre token generation Lambda trigger](#), you must configure your user pool with a feature plan other than *Lite* and update your Lambda trigger configuration to use event version 2.
- Except for [Custom sender Lambda triggers](#), Amazon Cognito invokes Lambda functions synchronously. When Amazon Cognito calls your Lambda function, it must respond within 5 seconds. If it doesn't and if the call can be retried, Amazon Cognito retries the call. After three unsuccessful attempts, the function times out. You can't change this five-second timeout value. For more information, see [Lambda programming model](#) in the AWS Lambda Developer Guide.

Amazon Cognito doesn't retry function calls that return an [Invoke error](#) with an HTTP status code of 500-599. These codes indicate a configuration issue that leaves Lambda unable to launch the function. For more information, see [Error handling and automatic retries in AWS Lambda](#).

- You can't declare a function version in your Lambda trigger configuration. Amazon Cognito user pools invoke the latest version of your function by default. However, you can associate a function version with an alias and set your trigger LambdaArn to the alias ARN in a [CreateUserPool](#) or

[UpdateUserPool](#) API request. This option isn't available in the AWS Management Console. For more information about aliases, see [Lambda function aliases](#) in the *AWS Lambda Developer Guide*.

- If you delete a 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**.
- If your Lambda function doesn't return the request and response parameters to Amazon Cognito, or returns an error, the authentication event doesn't succeed. You can return an error in your function to prevent a user's sign-up, authentication, token generation, or any other stage of their authentication flow that invokes Lambda trigger.

Managed login returns errors that Lambda triggers generate as error text above the sign-in prompt. The Amazon Cognito user pools API returns trigger errors in the format `[trigger] failed with error [error text from response]`. As a best practice, only generate errors in your Lambda functions that you want your users to see. Use output methods like `print()` to log any sensitive or debugging information to CloudWatch Logs. For an example, see [Pre sign-up example: Deny sign-up if user name has fewer than five characters](#).

- You can add a Lambda function in another AWS account as a trigger for your user pool. You must add cross-account triggers with the [CreateUserPool](#) and [UpdateUserPool](#) API operations, or their equivalents in AWS CloudFormation and the AWS CLI. You can't add cross-account functions in the AWS Management Console.
- When you add a Lambda trigger in the Amazon Cognito console, Amazon Cognito adds a resource-based policy to your function that permits your user pool to invoke the function. When you create a Lambda trigger outside of the Amazon Cognito console, including a cross-account function, you must add permissions to the resource-based policy of the Lambda function. Your added permissions must allow Amazon Cognito to invoke the function on behalf of your user pool. You can [add permissions from the Lambda Console](#) or use the Lambda [AddPermission](#) API operation.

Example Lambda Resource-Based Policy

The following example Lambda resource-based policy grants Amazon Cognito a limited ability to invoke a Lambda function. Amazon Cognito can only invoke the function when it does so on behalf of both the user pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

{

```
"Version": "2012-10-17",
"Id": "default",
"Statement": [
    {
        "Sid": "lambda-allow-cognito",
        "Effect": "Allow",
        "Principal": {
            "Service": "cognito-idp.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": "<your Lambda function ARN>",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "<your account number>"
            },
            "ArnLike": {
                "AWS:SourceArn": "<your user pool ARN>"
            }
        }
    }
]
```

Adding a user pool Lambda trigger

To add a user pool Lambda trigger with the console

1. Use the [Lambda console](#) to create a Lambda function. For more information on Lambda functions, see the [AWS Lambda Developer Guide](#).
2. Go to the [Amazon Cognito console](#), and then choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Extensions** menu and locate **Lambda triggers**.
5. Choose **Add a Lambda trigger**.
6. Select a Lambda trigger **Category** based on the stage of authentication that you want to customize.
7. Select **Assign Lambda function** and select a function in the same AWS Region as your user pool.

Note

If your AWS Identity and Access Management (IAM) credentials have permission to update the Lambda function, Amazon Cognito adds a Lambda resource-based policy. With this policy, Amazon Cognito can invoke the function that you select. If the signed-in credentials do not have sufficient IAM permissions, you must update the resource-based policy separately. For more information, see [the section called “Important considerations”](#).

8. Choose **Save changes**.
9. You can use CloudWatch in the Lambda console to log your Lambda function . For more information, see [Accessing CloudWatch Logs for Lambda](#).

User pool Lambda trigger event

Amazon Cognito passes event information to your Lambda function. The Lambda function 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 [Connecting Lambda triggers to user pool functional operations](#).

region

The AWS Region as an AWSRegion instance.

userPoolId

The ID of the user pool.

userName

The current user's username.

callerContext

Metadata about the request and the code environment. It contains the fields **awsSdkVersion** and **clientId**.

awsSdkVersion

The version of the AWS SDK that generated the request.

clientId

The ID of the user pool app client.

request

Details of your user's API request. It includes the following fields, and any request parameters that are particular to the trigger. For example, an event that Amazon Cognito sends to a pre-authentication trigger will also contain a `userNotFound` parameter. You can process the value

of this parameter to take a custom action when your user tries to sign in with an unregistered username.

userAttributes

One or more key-value pairs of user attribute names and values, for example "email": "john@example.com".

response

This parameter doesn't contain any information in the original request. Your Lambda function must return the entire event to Amazon Cognito, and add any return parameters to the response. To see what return parameters your function can include, refer to the documentation for the trigger that you want to use.

Connecting API operations to Lambda triggers

The following sections describe the Lambda triggers that Amazon Cognito invokes from the activity in your user pool.

When your app signs in users through the Amazon Cognito user pools API, managed login, or user pool endpoints, Amazon Cognito invokes your Lambda functions based on the session context. For more information about the Amazon Cognito user pools API and user pool endpoints, see [Understanding API, OIDC, and managed login pages authentication](#). The tables in the sections that follow describe events that cause Amazon Cognito to invoke a function, and the triggerSource string that Amazon Cognito includes in the request.

Topics

- [Lambda triggers in the Amazon Cognito API](#)
- [Lambda triggers for Amazon Cognito local users in managed login](#)
- [Lambda triggers for federated users](#)

Lambda triggers in the Amazon Cognito API

The following table describes the source strings for the Lambda triggers that Amazon Cognito can invoke when your app creates, signs in, or updates a local user.

Local user trigger sources in the Amazon Cognito API

API operation	Lambda trigger	Trigger source
AdminCreateUser	Pre sign-up	PreSignUp_AdminCreateUser
	Pre token generation	TokenGeneration_NewPasswordChallenge
	Custom message	CustomMessage_AdminCreateUser
	Custom email sender	CustomEmailSender_AdminCreateUser
	Custom SMS sender	CustomSMSSender_AdminCreateUser
SignUp	Pre sign-up	PreSignUp_SignUp
	Custom message	CustomMessage_SignUp
	Custom email sender	CustomEmailSender_SignUp
	Custom SMS sender	CustomSMSSender_SignUp
ConfirmSignUp	Post confirmation	PostConfirmation_ConfirmSignUp
AdminConfirmSignUp		
InitiateAuth	Pre authentication	PreAuthentication_Authentication
AdminInitiateAuth	Define auth challenge	DefineAuthChallenge_Authentication
	Create auth challenge	CreateAuthChallenge_Authentication

API operation	Lambda trigger	Trigger source
GetUser	Pre token generation	TokenGeneration_Authentication TokenGeneration_AuthenticateDevice TokenGeneration_RefreshTokens
	Migrate user	UserMigration_Authentication
	Custom message	CustomMessage_Authentication
ForgotPassword	Custom email sender	CustomEmailSender_AccountTakeOverNotification CustomEmailSender_Authentication
	Custom SMS sender	CustomSMSSender_Authentication
	Migrate user	UserMigration_ForgotPassword
GetaptchaChallenge	Custom message	CustomMessage_ForgotPassword
	Custom email sender	CustomEmailSender_ForgotPassword
	Custom SMS sender	CustomSMSSender_ForgotPassword

API operation	Lambda trigger	Trigger source
ConfirmForgotPassword	Post confirmation	PostConfirmation_ConfirmForgotPassword
UpdateUserAttributes AdminUpdateUserAttributes	Custom message	CustomMessage_UpdateUserAttribute
	Custom email sender	CustomEmailSender_UpdateUserAttribute
	Custom SMS sender	CustomSMSender_UpdateUserAttribute
VerifyUserAttributes	Custom message	CustomMessage_VerifyUserAttribute
	Custom email sender	CustomEmailSender_VerifyUserAttribute
	Custom SMS sender	CustomSMSender_VerifyUserAttribute

Lambda triggers for Amazon Cognito local users in managed login

The following table describes the source strings for the Lambda triggers that Amazon Cognito can invoke when a local user signs in to your user pool with managed login.

Local user trigger sources in managed login

Managed login URI	Lambda trigger	Trigger source
/signup	Pre sign-up	PreSignUp_SignUp
	Custom message	CustomMessage_SignUp
	Custom email sender	CustomEmailSender_SignUp

Managed login URI	Lambda trigger	Trigger source
	Custom SMS sender	CustomSMSender_SignUp
/confirmuser	Post confirmation	PostConfirmation_ConfirmSignUp
/login	Pre authentication	PreAuthentication_Authentication
	Define auth challenge	DefineAuthChallenge_Authentication
	Create auth challenge	CreateAuthChallenge_Authentication
	Pre token generation	TokenGeneration_Authentication
		TokenGeneration_AuthenticateDevice
		TokenGeneration_RefreshTokens
	Migrate user	UserMigration_Authentication
	Custom message	CustomMessage_Authentication
	Custom email sender	CustomEmailSender_AccountTakeOverNotification
		CustomEmailSender_Authentication

Managed login URI	Lambda trigger	Trigger source
	Custom SMS sender	CustomSMSender_Authentication
/forgotpassword	Migrate user	UserMigration_ForgotPassword
	Custom message	CustomMessage_ForgotPassword
	Custom email sender	CustomEmailSender_ForgotPassword
	Custom SMS sender	CustomSMSender_ForgotPassword
/confirmforgotpassword	Post confirmation	PostConfirmation_ConfirmForgotPassword

Lambda triggers for federated users

You can use the following Lambda triggers to customize your user pool workflows for users who sign in with a federated provider.

 **Note**

Federated users can use managed login to sign in, or you can generate a request to the [Authorize endpoint](#) that silently redirects them to their identity provider sign-in page. You can't sign in federated users with the Amazon Cognito user pools API.

Federated user trigger sources

Sign-in event	Lambda trigger	Trigger source
First sign-in	Pre sign-up	PreSignUp_ExternalProvider
	Post confirmation	PostConfirmation_ConfirmSignUp
	Pre token generation	TokenGeneration_HostedAuth
Subsequent sign-ins	Pre authentication	PreAuthentication_Authentication
	Post authentication	PostAuthentication_Authentication
	Pre token generation	TokenGeneration_HostedAuth

Federated sign-in does not invoke any [Custom authentication challenge Lambda triggers](#), [Migrate user Lambda trigger](#), [Custom message Lambda trigger](#), or [Custom sender Lambda triggers](#) in your user pool.

Connecting Lambda triggers to user pool functional operations

Each Lambda trigger serves a functional role in your user pool. For example, a trigger can modify your sign-up flow, or add a custom authentication challenge. The event that Amazon Cognito sends to a Lambda function can reflect one of multiple actions that make up that functional role. For example, Amazon Cognito invokes a pre sign-up trigger when your user signs up, and when you create a user. Each of these different cases for the same functional role has its own `triggerSource` value. Your Lambda function can process incoming events differently based on the operation that invoked it.

Amazon Cognito also invokes all assigned functions when an event corresponds to a trigger source. For example, when a user signs in to a user pool where you assigned migrate user and pre authentication triggers, they activate both.

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

Trigger	triggerSource value	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.
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	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	Event
Pre token generation	TokenGeneration_HostedAuth	Amazon Cognito authenticates the user from your managed login sign-in page.
Pre token generation	TokenGeneration_Authentication	User authentication flows complete.
Pre token generation	TokenGeneration_NewPasswordChallenge	Admin creates the user. Amazon Cognito invokes this when the user must change a temporary password.
Pre token generation	TokenGeneration_AuthenticateDevice	End of the authentication of a user device.
Pre token generation	TokenGeneration_RefreshTokens	User tries to refresh the identity and access tokens.

Migrate user triggers

Trigger	triggerSource value	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	Event
Custom message	CustomMessage_SignUp	Custom message when a user signs up in your user pool.

Trigger	triggerSource value	Event
Custom message	CustomMessage_AdminCreateUser	Custom message when you create a user as an administrator and Amazon Cognito sends them a temporary password.
Custom message	CustomMessage_ResendCode	Custom message when your existing user requests a new confirmation code.
Custom message	CustomMessage_ForgotPassword	Custom message when your user requests a password reset.
Custom message	CustomMessage_UpdateUserAttribute	Custom message when a user changes their email address or phone number and Amazon Cognito sends a verification code.
Custom message	CustomMessage_VerifyUserAttribute	Custom message when a user adds an email address or phone number and Amazon Cognito sends a verification code.
Custom message	CustomMessage_Authentication	Custom message when a user who has configured SMS MFA signs in.

Pre sign-up Lambda trigger

You might want to customize the sign-up process in user pools that have self-service sign-up options. Some common uses of the pre sign-up trigger are to perform custom analysis and recording of new users, apply security and governance standards, or link users from a third-

party IdP to a [consolidated user profile](#). You might also have trusted users who aren't required to undergo [verification and confirmation](#).

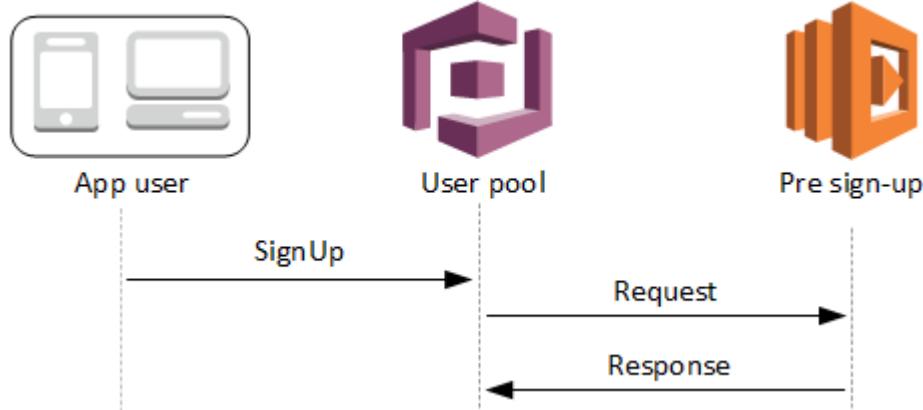
Shortly before Amazon Cognito signs up a new [local](#) or [federated](#) user, it activates the pre sign-up Lambda function. As part of the sign-up process, you can use this function to analyze the sign-in event with custom logic, and modify or deny the new user.

Topics

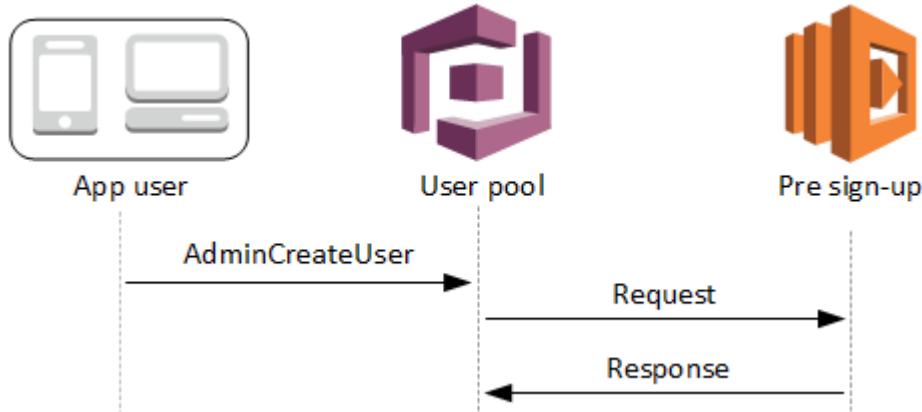
- [Pre sign-up Lambda flows](#)
- [Pre sign-up Lambda trigger parameters](#)
- [Sign-up tutorials](#)
- [Pre sign-up example: Auto-confirm users from a registered domain](#)
- [Pre sign-up example: Auto-confirm and auto-verify all users](#)
- [Pre sign-up example: Deny sign-up if user name has fewer than five characters](#)

Pre sign-up Lambda flows

Client sign-up flow



Server sign-up flow



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

Pre sign-up Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

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 key-value pairs with user attribute data that your app passed to Amazon Cognito in the request to create a new user. Send this information to your Lambda function in the ValidationData parameter of your [AdminCreateUser](#) or [SignUp](#) API request.

Amazon Cognito doesn't set your ValidationData data as attributes of the user that you create. ValidationData is temporary user information that you supply for the purposes of your pre sign-up Lambda trigger.

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 function 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 address 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 address when autoVerifyEmail is set. If an alias with that email address already exists, the alias will be moved to the new user and the previous user's email address will be marked as unverified.

For more information, see [Customizing sign-in attributes](#).

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 [Customizing sign-in attributes](#).

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

You can use the pre sign-up Lambda trigger to add custom logic that validates new users who sign up for your user pool. This is a sample JavaScript program that shows how to sign up a new user. It invokes 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 to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is 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

```
const handler = async (event) => {
  // Confirm the user
  event.response.autoConfirmUser = true;
  // Set the email as verified if it is in the request
  if (Object.hasOwnProperty(event.request.userAttributes, "email")) {
    event.response.autoVerifyEmail = true;
  }

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

  return event;
};

export { handler };
```

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 to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is 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": {}  
}
```

Pre sign-up example: Deny sign-up if user name has fewer than five characters

This example checks the length of the user name in a sign-up request. The example returns an error if the user has entered a name less than five characters long.

Node.js

```
exports.handler = (event, context, callback) => {  
  // Impose a condition that the minimum length of the username is 5 is imposed on  
  // all user pools.  
  if (event.userName.length < 5) {  
    var error = new Error("Cannot register users with username less than the  
    minimum length of 5");  
    // Return error to Amazon Cognito  
    callback(error, event);  
  }  
  // Return to Amazon Cognito  
  callback(null, event);  
};
```

Python

```
def lambda_handler(event, context):  
    if len(event['userName']) < 5:
```

```
        raise Exception("Cannot register users with username less than the minimum  
length of 5")  
    # Return to Amazon Cognito  
    return event
```

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

JSON

```
{  
    "user_name": "rroe",  
    "response": {}  
}
```

Post confirmation Lambda trigger

Amazon Cognito invokes this trigger after a signed-up user confirms their user account. In your post confirmation Lambda function, you can send custom messages or add custom API requests. For example, you can query an external system and populate additional attributes to the user. Amazon Cognito invokes this trigger only for user who sign up in your user pool, not for user accounts that you create with your administrator credentials.

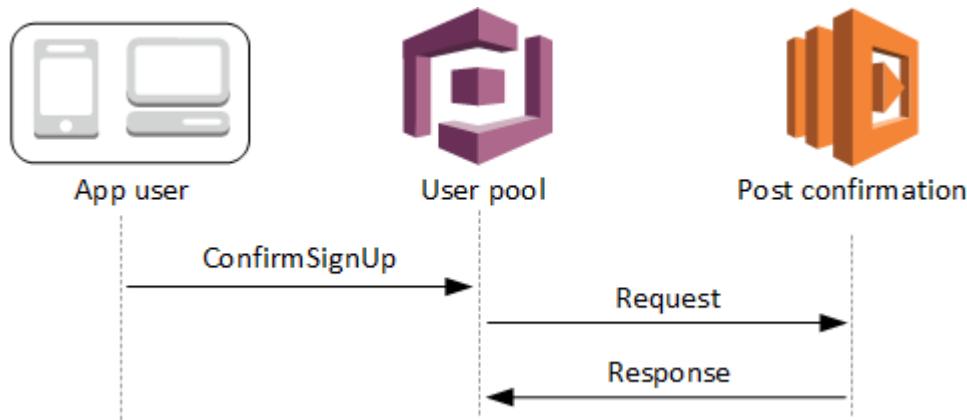
The request contains the current attributes for the confirmed user.

Topics

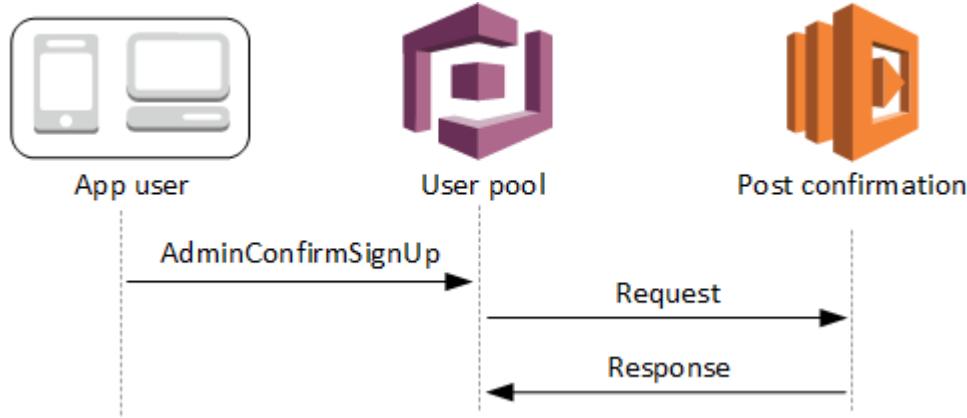
- [Post confirmation Lambda flows](#)
- [Post confirmation Lambda trigger parameters](#)
- [User confirmation tutorials](#)
- [Post confirmation example](#)

Post confirmation Lambda flows

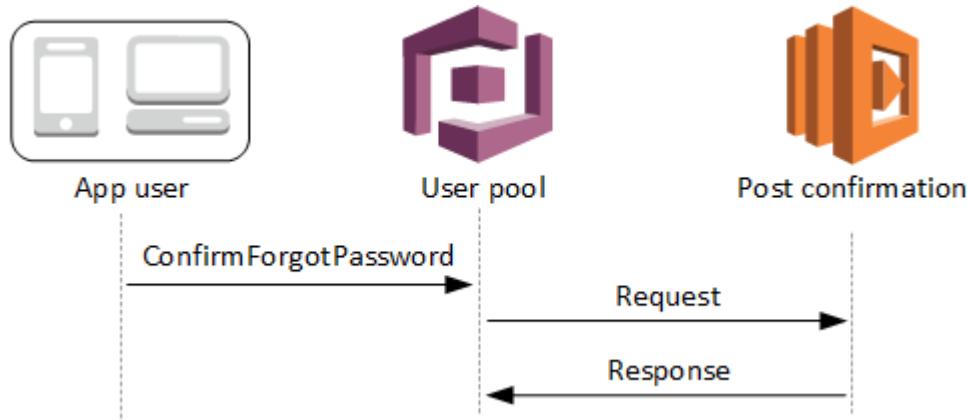
Client confirm sign-up flow



Server confirm sign-up flow



Confirm forgot password flow



Post confirmation Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

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

```
// Import required AWS SDK clients and commands for Node.js. Note that this requires
// the `@aws-sdk/client-ses` module to be either bundled with this code or included
// as a Lambda layer.
import { SES, SendEmailCommand } from "@aws-sdk/client-ses";
const ses = new SES();

const handler = async (event) => {
  if (event.request.userAttributes.email) {
    await sendTheEmail(
      event.request.userAttributes.email,
      `Congratulations ${event.userName}, you have been confirmed.`,
    );
  }
  return event;
};

const sendTheEmail = async (to, body) => {
  const 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>",
};

try {
    await ses.send(new SendEmailCommand(eParams));
} catch (err) {
    console.log(err);
}
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is 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 so that you can create custom validation that performs preparatory actions. For example, you can deny the authentication request or record session data to an external system.

Note

This Lambda trigger doesn't activate when a user doesn't exist, or already has an existing session in your user pool. If the `PreventUserExistenceErrors` setting of a user pool app client is set to `ENABLED`, then the Lambda trigger will activate.

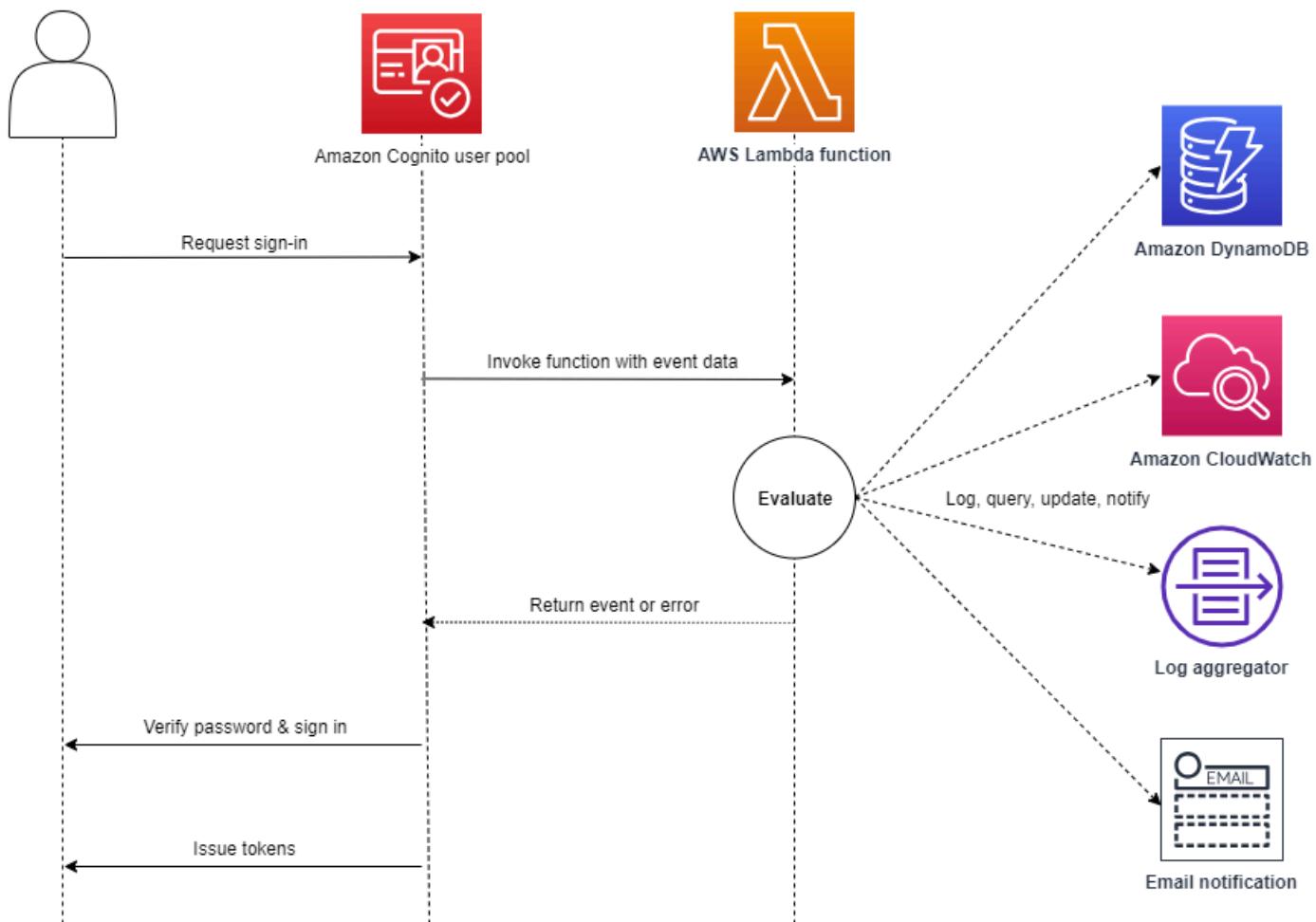
Topics

- [Authentication flow overview](#)
- [Pre authentication Lambda trigger parameters](#)
- [Pre authentication example](#)

Authentication flow overview

Amazon Cognito pre authentication trigger

Evaluate and authorize user sign-in



The request includes client validation data from the `ClientMetadata` values that your app passes to the user pool `InitiateAuth` and `AdminInitiateAuth` API operations.

For more information, see [An example authentication session](#).

Pre authentication Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

JSON

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

Pre authentication request parameters

userAttributes

One or more name-value pairs that represent user attributes.

userNotFound

When you set `PreventUserExistenceErrors` to `ENABLED` for your user pool client, Amazon Cognito populates this Boolean.

validationData

One or more key-value pairs that contain the validation data in the user's sign-in request. To pass this data to your Lambda function, use the `ClientMetadata` parameter in the [InitiateAuth](#) and [AdminInitiateAuth](#) API actions.

Pre authentication response parameters

Amazon Cognito does not expect any additional return information in the response. Your function can return an error to reject the sign-in attempt, or use API operations to query and modify your resources.

Pre authentication example

This example function prevents users from signing in to your user pool with a specific app client. Because the pre authentication Lambda function doesn't invoke when your user has an existing session, this function only prevents *new* sessions with the app client ID that you want to block.

Node.js

```
const handler = async (event) => {
  if (
    event.callerContext.clientId === "user-pool-app-client-id-to-be-blocked"
  ) {
    throw new Error("Cannot authenticate users from this user pool app client");
  }

  return event;
};

export { handler };
```

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 to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is 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

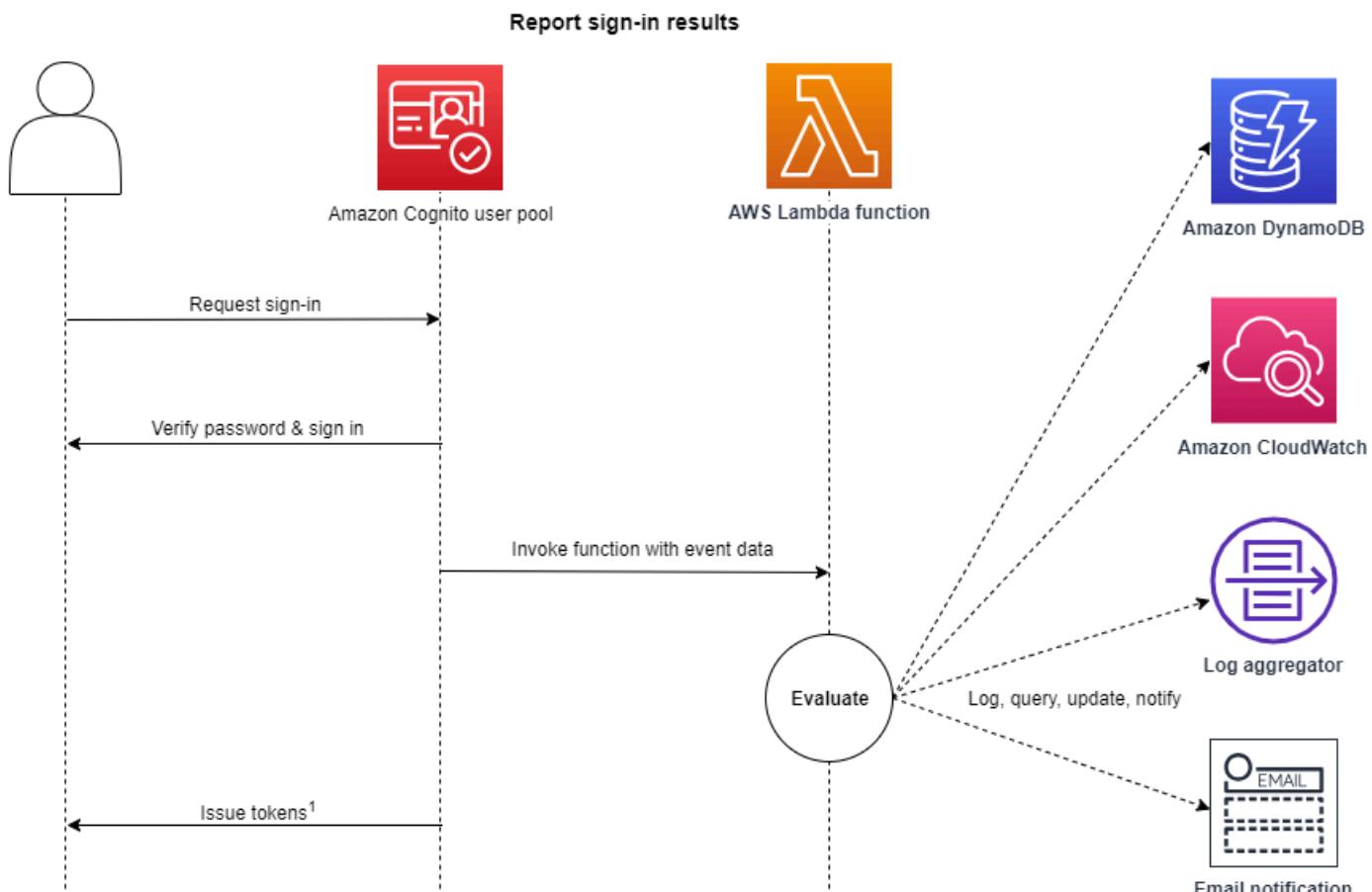
The post authentication trigger doesn't change the authentication flow for a user. Amazon Cognito invokes this Lambda after authentication is complete and a user has received tokens. Add a post authentication trigger when you want to add custom post-processing of authentication events, for example logging or user profile adjustments that will be reflected on the next sign-in.

Topics

- [Authentication flow overview](#)
- [Post authentication Lambda trigger parameters](#)
- [Authentication tutorials](#)
- [Post authentication example](#)

Authentication flow overview

Amazon Cognito post authentication trigger



¹ This trigger doesn't have any effect on sign-in outcomes or token contents.

For more information, see [An example authentication session](#).

Post authentication Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

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. Amazon Cognito only sets this flag if the remembered devices value of the user pool is 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. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the [AdminRespondToAuthChallenge](#) and [RespondToAuthChallenge](#) API actions. Amazon Cognito doesn't include data from the ClientMetadata parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations in the request that it passes to the post authentication function.

Post authentication response parameters

Amazon Cognito doesn't expect any additional return information in the response. Your function can use API operations to query and modify your resources, or record event metadata to an external system.

Authentication tutorials

Immediately after Amazon Cognito signs in a user, it activates the post authentication Lambda function. 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

```
const handler = async (event) => {
    // Send post authentication data to Amazon 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 event;
};

export { handler };
```

Python

```
import os
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 to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is 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

As you build out your authentication flows for your Amazon Cognito user pool, you might find that you want to extend your authentication model beyond the built-in flows. One common use case for the custom challenge triggers is to implement additional security checks beyond username, password, and multi-factor authentication (MFA). A custom challenge is any question and response you can generate in a Lambda-supported programming language. For example, you might want to require users to solve a CAPTCHA or answer a security question before being allowed to authenticate. Another potential need is to integrate with specialized authentication factors or devices. Or you might have already developed software that authenticates users with a hardware security key or a biometric device. The definition of authentication success for a custom challenge is whatever answer your Lambda function accepts as correct: a fixed string, for example, or a satisfactory response from an external API.

You can start authentication with your custom challenge and control the authentication process entirely, or you can perform username-password authentication before your application receives your custom challenge.

The custom authentication challenge Lambda trigger:

Defines

Initiates a challenge sequence. Determines whether you want to initiate a new challenge, mark authentication as complete, or halt the authentication attempt.

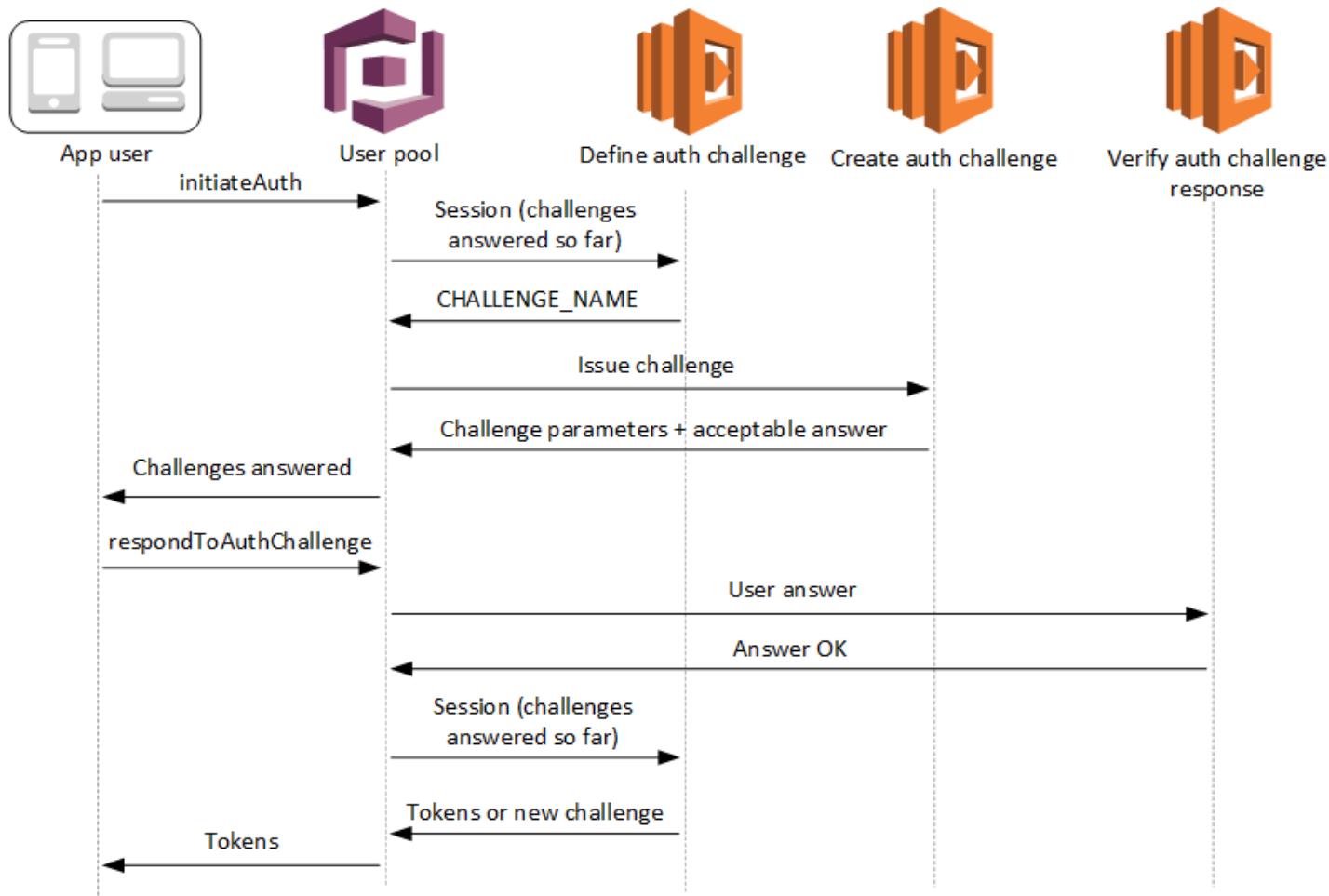
Creates

Issues the question to your application that the user must answer. This function might present a security question or a link to a CAPTCHA that your application should display to your user.

Verifies

Knows the expected answer and compares it to the answer your application provides in the challenge response. The function might call the API of your CAPTCHA service to retrieve the expected results of your user's attempted solution.

These three Lambda functions chain together to present an authentication mechanism that is completely within your control and of your own design. Because custom authentication requires application logic in your client and in the Lambda functions, you can't process custom authentication within managed login. This authentication system requires additional developer effort. Your application must perform the authentication flow with the user pools API and handle the resulting challenge with a custom-built login interface that renders the question at the center of the custom authentication challenge.



For more information about implementing custom authentication, see [Custom authentication flow and challenges](#)

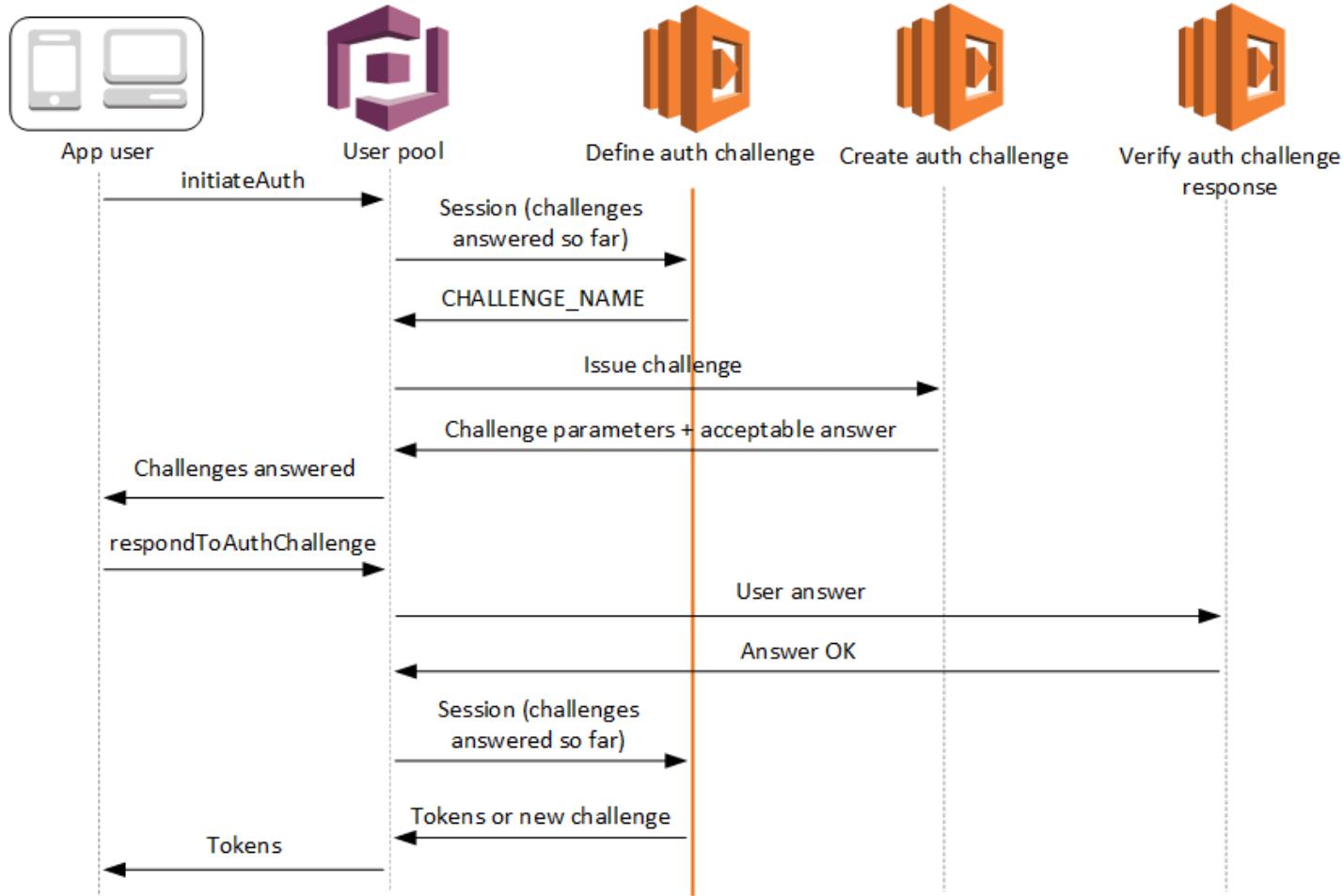
Authentication between the API operations [InitiateAuth](#) or [AdminInitiateAuth](#), and [RespondToAuthChallenge](#) or [AdminRespondToAuthChallenge](#). In this flow, a user authenticates by answering successive challenges until authentication either fails or the user is issued tokens. A challenge response might be a new challenge. In this case, your application responds as many times as necessary to new challenges. Successful authentication happens when the define auth challenge function analyzes the results so far, determines all challenges have been answered, and returns `IssueTokens`.

Topics

- [Define Auth challenge Lambda trigger](#)
- [Create Auth challenge Lambda trigger](#)
- [Verify Auth challenge response Lambda trigger](#)

Define Auth challenge Lambda trigger

The define auth challenge trigger is a Lambda function that maintains the challenge sequence in a custom authentication flow. It declares success or failure of the challenge sequence, and sets the next challenge if the sequence isn't yet complete.



Define auth challenge

Amazon Cognito invokes this trigger to initiate the [custom authentication flow](#).

The request for this Lambda trigger contains `session`. The `session` parameter 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 `session` array stores challenge details (`ChallengeResult`) in chronological order. The challenge `session[0]` represents the first challenge that the user receives.

You can have Amazon Cognito verify user passwords before it issues your custom challenges. Any Lambda triggers associated in the Authentication category of [request-rate quotas](#) will run when you perform SRP authentication in a custom challenge flow. Here is an overview of the process:

1. Your app initiates sign-in by calling `InitiateAuth` or `AdminInitiateAuth` with the `AuthParameters` map. Parameters must include `CHALLENGE_NAME: SRP_A`, and values for `SRP_A` and `USERNAME`.
2. Amazon Cognito invokes your define auth challenge Lambda trigger 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, Amazon Cognito invokes your Lambda function again with a new session containing `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`.
5. To initiate your custom challenges, your Lambda function responds 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.

The following is an example of a starting `InitiateAuth` request that precedes custom authentication with an SRP flow.

```
{  
    "AuthFlow": "CUSTOM_AUTH",  
    "ClientId": "1example23456789",  
    "AuthParameters": {  
        "CHALLENGE_NAME": "SRP_A",  
        "USERNAME": "testuser",  
        "SRP_A": "[SRP_A]",  
        "SECRET_HASH": "[secret hash]"  
    }  
}
```

Topics

- [Define Auth challenge Lambda trigger parameters](#)
- [Define Auth challenge example](#)

Define Auth challenge Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

JSON

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

Define Auth challenge request parameters

When Amazon Cognito invokes your Lambda function, Amazon Cognito provides the following parameters:

userAttributes

One or more name-value pairs that represent user attributes.

userNotFound

A Boolean that Amazon Cognito populates when `PreventUserExistenceErrors` is set to `ENABLED` for your user pool client. A value of `true` means that the user id

(username, email address, and other details) did not match any existing users. When `PreventUserExistenceErrors` is set to `ENABLED`, the service doesn't inform the app of nonexistent users. We recommend that your Lambda functions maintain the same user experience and account for latency. This way, the caller can't detect different behavior when the user exists or doesn't exist.

session

An array of `ChallengeResult` elements. Each contains the following elements:

challengeName

One of the following challenge types: `CUSTOM_CHALLENGE`, `SRP_A`, `PASSWORD_VERIFIER`, `SMS_MFA`, `EMAIL OTP`, `SOFTWARE_TOKEN_MFA`, `DEVICE_SRP_AUTH`, `DEVICE_PASSWORD_VERIFIER`, or `ADMIN_NO_SRP_AUTH`.

When your `define auth challenge` function issues a `PASSWORD_VERIFIER` challenge for a user who has set up multifactor authentication, Amazon Cognito follows it up with an `SMS_MFA`, `EMAIL OTP`, or `SOFTWARE_TOKEN_MFA` challenge. These are the prompts for a multi-factor authentication code. In your function, include handling for input events from `SMS_MFA`, `EMAIL OTP`, and `SOFTWARE_TOKEN_MFA` challenges. You don't need to invoke any MFA challenges in your `define auth challenge` function.

Important

When your function is determining whether a user has successfully authenticated and you should issue them tokens, always check `challengeName` in your `define auth challenge` function and verify that it matches the expected value.

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. To pass this data to your Lambda function,

you can use the `ClientMetadata` parameter in the [AdminRespondToAuthChallenge](#) and [RespondToAuthChallenge](#) API operations. The request that invokes the `defineAuthChallenge` function doesn't include data passed in the `ClientMetadata` parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations.

Define Auth challenge response parameters

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

challengeName

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

issueTokens

If you determine that the user has completed the authentication challenges sufficiently, set to `true`. If the user has not met the challenges sufficiently, set to `false`.

failAuthentication

If you want to end the current authentication process, set to `true`. To continue the current authentication process, set to `false`.

Define Auth challenge example

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

Node.js

```
const handler = async (event) => {
  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 &&
```

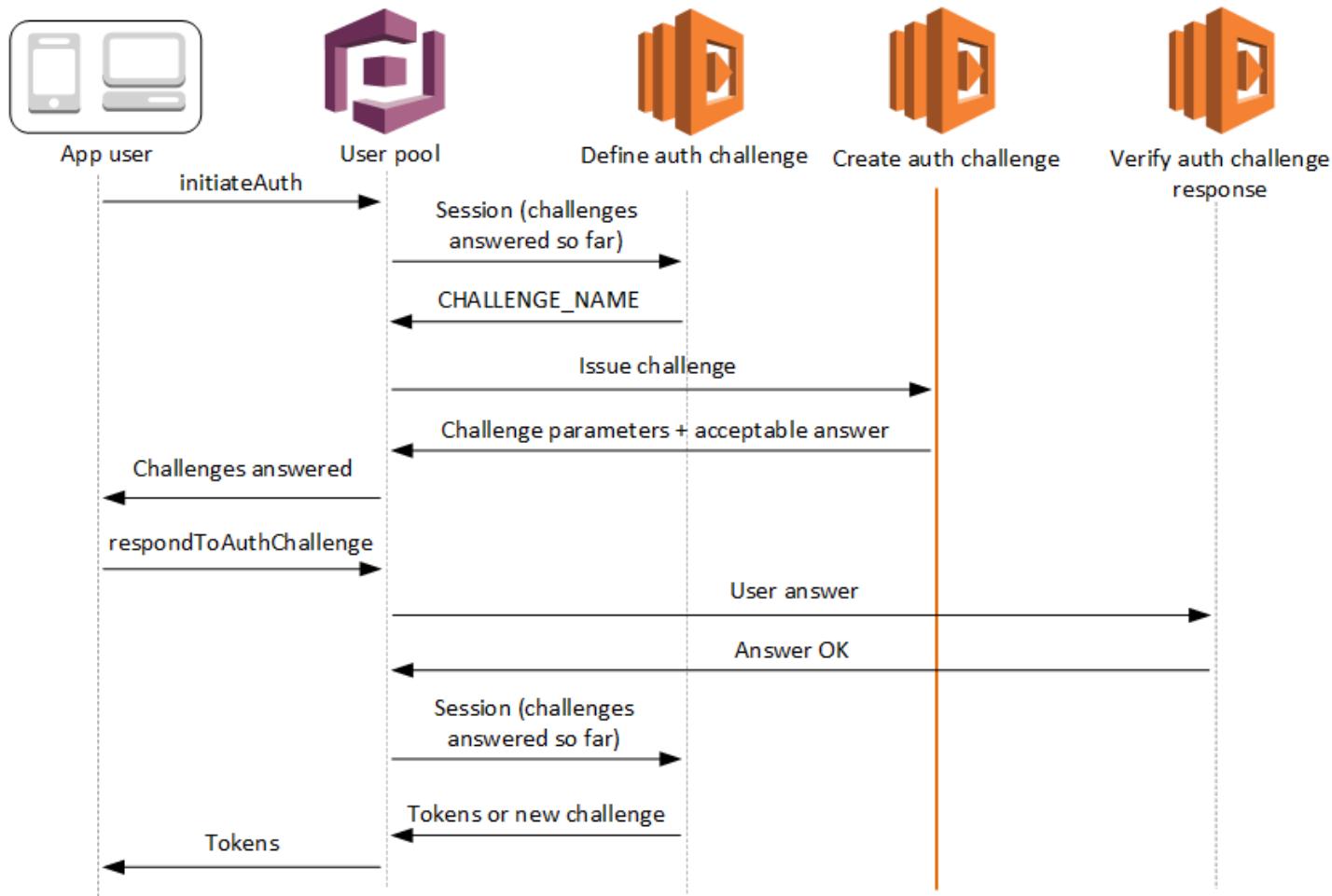
```
event.request.session[1].challengeName === "PASSWORD_VERIFIER" &&
event.request.session[1].challengeResult === true
) {
  event.response.issueTokens = false;
  event.response.failAuthentication = false;
  event.response.challengeName = "CUSTOM_CHALLENGE";
} else if (
  event.request.session.length === 3 &&
  event.request.session[2].challengeName === "CUSTOM_CHALLENGE" &&
  event.request.session[2].challengeResult === true
) {
  event.response.issueTokens = false;
  event.response.failAuthentication = false;
  event.response.challengeName = "CUSTOM_CHALLENGE";
} else if (
  event.request.session.length === 4 &&
  event.request.session[3].challengeName === "CUSTOM_CHALLENGE" &&
  event.request.session[3].challengeResult === true
) {
  event.response.issueTokens = true;
  event.response.failAuthentication = false;
} else {
  event.response.issueTokens = false;
  event.response.failAuthentication = true;
}

return event;
};

export { handler };
```

Create Auth challenge Lambda trigger

The create auth challenge trigger is a Lambda function that has the details of each challenge declared by the define auth challenge trigger. It processes the challenge name declared by the define auth challenge trigger and returns a `publicChallengeParameters` that your application must present to the user. This function then provides your user pool with the answer to the `challenge`, `privateChallengeParameters`, that your user pool passes to the verify auth challenge trigger. Where your define auth challenge trigger manages the challenge sequence, your create auth challenge trigger manages the challenge contents.



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](#)
- [Create Auth challenge example](#)

Create Auth challenge Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

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 use the `ClientMetadata` parameter in the [AdminRespondToAuthChallenge](#) and [RespondToAuthChallenge](#) API actions to pass this data to your Lambda function. The request that invokes the create auth challenge function does not include data passed in the `ClientMetadata` parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations.

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 present the challenge to the user accurately.

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

```
const handler = async (event) => {
  if (event.request.challengeName !== "CUSTOM_CHALLENGE") {
    return event;
  }

  if (event.request.session.length === 2) {
    event.response.publicChallengeParameters = {};
    event.response.privateChallengeParameters = {};
    event.response.publicChallengeParameters.captchaUrl = "url/123.jpg";
    event.response.privateChallengeParameters.answer = "5";
  }

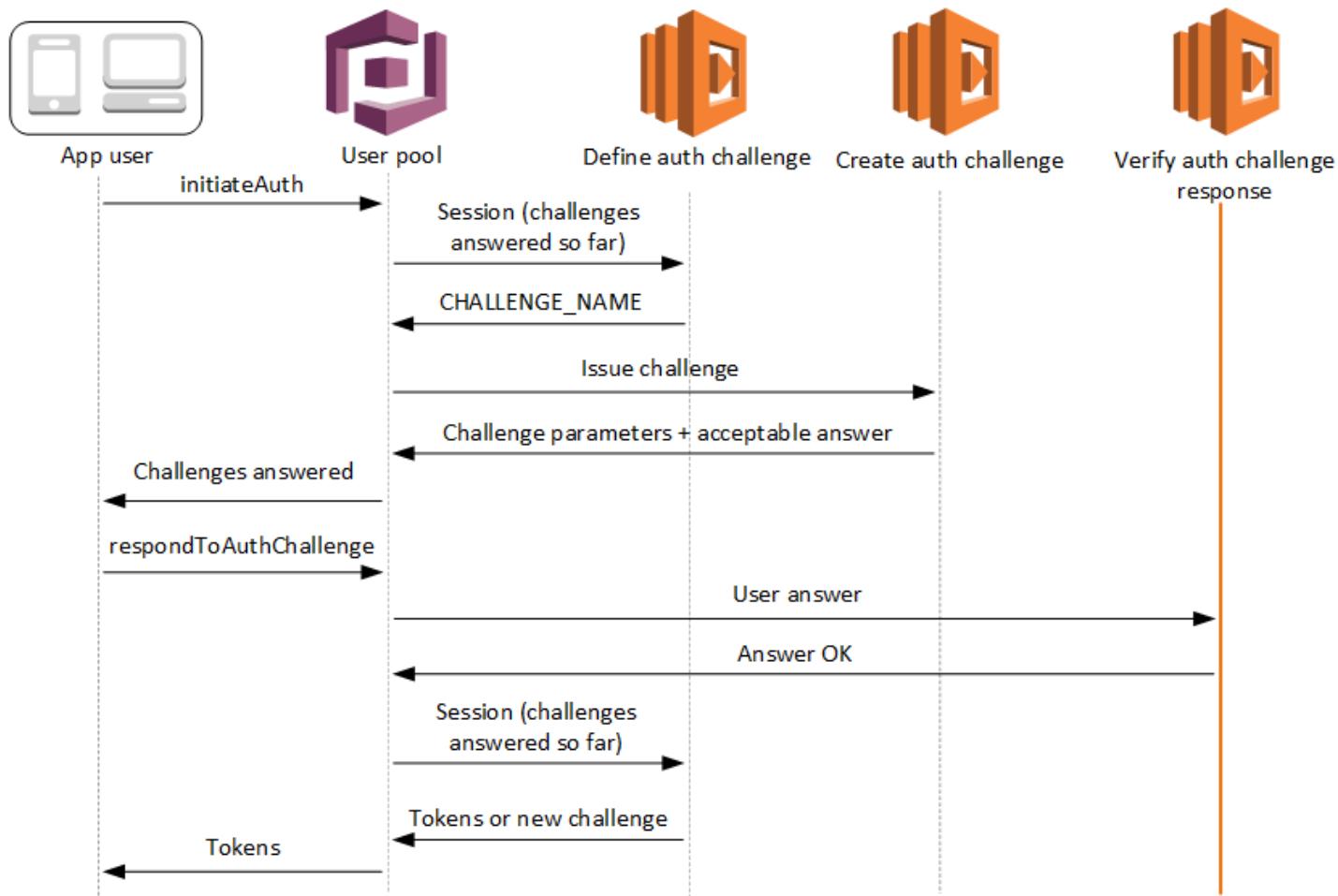
  if (event.request.session.length === 3) {
    event.response.publicChallengeParameters = {};
    event.response.privateChallengeParameters = {};
    event.response.publicChallengeParameters.securityQuestion =
      "Who is your favorite team mascot?";
    event.response.privateChallengeParameters.answer = "Peccy";
  }

  return event;
};
```

```
export { handler };
```

Verify Auth challenge response Lambda trigger

The verify auth challenge trigger is a Lambda function that compares a user's provided response to a known answer. This function tells your user pool whether the user answered the challenge correctly. When the verify auth challenge trigger responds with an `answerCorrect` of `true`, the authentication sequence can continue.



Verify auth challenge response

Amazon Cognito invokes this trigger to verify if the response from the 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 Create Auth Challenge Lambda trigger returns

`privateChallengeParameters` values, and contains the expected response from the user. The `challengeAnswer` parameter contains the user's response for the challenge.

The response contains the `answerCorrect` attribute. If the user successfully completes the challenge, Amazon Cognito sets the attribute value to `true`. If the user doesn't successfully complete the challenge, Amazon Cognito sets the value to `false`.

The challenge loop repeats until the users answers all challenges.

Topics

- [Verify Auth challenge Lambda trigger parameters](#)
- [Verify Auth challenge response example](#)

Verify Auth challenge Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

JSON

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

Verify Auth challenge request parameters

userAttributes

This parameter contains one or more name-value pairs that represent user attributes.

userNotFound

When Amazon Cognito sets `PreventUserExistenceErrors` to `ENABLED` for your user pool client, Amazon Cognito populates this Boolean .

privateChallengeParameters

This parameter comes from the Create Auth Challenge trigger. To determine whether the user passed a challenge, Amazon Cognito compares the parameters against a user's `challengeAnswer`.

This parameter contains all of the information that is required to validate the user's response to the challenge. That information includes the question that Amazon Cognito presents to the user (`publicChallengeParameters`), and the valid answers for the question (`privateChallengeParameters`). Only the Verify Auth Challenge Response Lambda trigger uses this parameter.

challengeAnswer

This parameter value is the answer from the user's response to the challenge.

clientMetadata

This parameter contains one or more key-value pairs that you can provide as custom input to the Lambda function for the verify auth challenge trigger. To pass this data to your Lambda function, use the `ClientMetadata` parameter in the [AdminRespondToAuthChallenge](#) and [RespondToAuthChallenge](#) API operations. Amazon Cognito doesn't include data from the `ClientMetadata` parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations in the request that it passes to the verify auth challenge function.

Verify Auth challenge response parameters

answerCorrect

If the user successfully completes the challenge, Amazon Cognito sets this parameter to `true`. If the user doesn't successfully complete the challenge, Amazon Cognito sets the parameter to `false`.

Verify Auth challenge response example

In this example, the Lambda function checks whether the user's response to a challenge matches the expected response. If the user's response matches the expected response, Amazon Cognito sets the `answerCorrect` parameter to `true`.

Node.js

```
const handler = async (event) => {
  if (
    event.request.privateChallengeParameters.answer ===
    event.request.challengeAnswer
  ) {
    event.response.answerCorrect = true;
  } else {
    event.response.answerCorrect = false;
  }

  return event;
};

export { handler };
```

Pre token generation Lambda trigger

Because Amazon Cognito invokes this trigger before token generation, you can customize the claims in user pool tokens. With the **Basic features** of the version one or `V1_0` pre token generation trigger event, you can customize the identity (ID) token. In user pools with the Essentials or Plus feature plan, you can generate the version 2 or `V2_0` trigger event with **access token customization**.

Amazon Cognito sends a `V1_0` event as a request to your function with data that it would write to the ID token. A `V2_0` event is a single request with the data that Amazon Cognito would write to both the identity and access tokens. To customize both tokens, you must update your function to use the most recent trigger version, and send data for both tokens in the same response.

This Lambda trigger can add, remove, and modify some claims in identity and access tokens before Amazon Cognito issues them to your app. To use this feature, associate a Lambda function from the Amazon Cognito user pools console or update your user pool `LambdaConfig` through the AWS Command Line Interface (AWS CLI).

Event versions

Your user pool can deliver different versions of a pre token generation trigger event to your Lambda function. A V1_0 trigger delivers the parameters for modification of ID tokens. A V2_0 trigger delivers parameters for the following.

1. The functions of a V1_0 trigger.
2. The ability to customize access tokens.
3. The ability to pass complex datatypes to ID and access token claim values:
 - String
 - Number
 - Boolean
 - Array of strings, numbers, booleans, or a combination of any of these
 - JSON

 **Note**

In the ID token, you can populate complex objects to the values of claims except for phone_number_verified, email_verified, updated_at, and address.

User pools deliver V1_0 events by default. To configure your user pool to send a V2_0 event, choose a **Trigger event version** of **Basic features + access token customization** when you configure your trigger in the Amazon Cognito console. You can also set the value of LambdaVersion in the [LambdaConfig](#) parameters in an [UpdateUserPool](#) or [CreateUserPool](#) API request. Additional costs apply to access token customization with V2_0 events. For more information, see [Amazon Cognito Pricing](#).

Excluded claims and scopes

Amazon Cognito limits the claims and scopes that you can add, modify, or suppress in access and identity tokens. If your Lambda function attempts to set a value for any of these claims, Amazon Cognito issues a token with the original claim value, if one was present in the request.

Shared claims

- acr

- amr
- at_hash
- auth_time
- azp
- exp
- iat
- iss
- jti
- nbf
- nonce
- origin_jti
- sub
- token_use

ID token claims

- identities
- aud
- cognito:username

Access token claims

- username
- client_id
- scope

 **Note**

You can change the scopes in an access token with `scopesToAdd` and `scopesToSuppress` response values, but you can't modify the `scope` claim directly. You can't add scopes that begin with `aws.cognito`, including the user pools reserved scope `aws.cognito.signin.user.admin`.

- device_key
- event_id
- version

You can't add or override claims with the following prefixes but you can suppress them, or prevent them from appearing in the token.

- dev:
- cognito:

The IAM role claims cognito:roles and cognito:preferred_role are linked to user pool groups by default. To suppress these claims, suppress cognito:groups in the claimsToSuppress object.

You can add an aud claim to access tokens, but its value must match the app client ID of the current session. You can derive the client ID in the request event from event.callerContext.clientId.

Customizing the identity token

With the pre token generation Lambda trigger, you can customize the content of an identity (ID) token from your user pool. The ID token provides user attributes from a trusted identity source for sign-in to a web or mobile app. For more information about ID tokens, see [Understanding the identity \(ID\) token](#).

The uses of the pre token generation Lambda trigger with an ID token include the following.

- Make a change at runtime to the IAM role that your user requests from an identity pool.
- Add user attributes from an external source.
- Add or replace existing user attribute values.
- Suppress disclosure of user attributes that, because of your user's authorized scopes and the read access to attributes that you granted to your app client, would otherwise be passed to your app.

Customizing the access token

With the pre token generation Lambda trigger, you can customize the content of an access token from your user pool. The access token authorizes users to retrieve information from access-

protected resources like Amazon Cognito token-authorized API operations and third-party APIs. Although you can generate access tokens for machine-to-machine (M2M) authorization with Amazon Cognito with a client credentials grant, M2M requests don't invoke the pre token generation trigger function and can't issue customized access tokens. For more information about access tokens, see [Understanding the access token](#).

The uses of the pre token generation Lambda trigger with an access token include the following.

- Add or suppress OAuth 2.0 scopes in the scope claim. For example, you can add scopes to an access token that resulted from Amazon Cognito user pools API authentication, which only assigns the scope `aws.cognito.signin.user.admin`.
- Change a user's membership in user pool groups.
- Add claims that aren't already present in an Amazon Cognito access token.
- Suppress disclosure of claims that would otherwise be passed to your app.

To support access customization in your user pool, you must configure the user pool to generate an updated version of the trigger request. Update your user pool as shown in the following procedure.

AWS Management Console

To support access token customization in a pre token generation Lambda trigger

1. Go to the [Amazon Cognito console](#), and then choose **User Pools**.
2. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Extensions** menu and locate **Lambda triggers**.
4. Add or edit a **Pre token generation** trigger.
5. Choose a Lambda function under **Assign Lambda function**.
6. Choose a **Trigger event version** of **Basic features + access token customization**. This setting updates the request parameters that Amazon Cognito sends to your function to include fields for access token customization.

User pools API

To support access token customization in a pre token generation Lambda trigger

Generate a [CreateUserPool](#) or [UpdateUserPool](#) API request. You must specify a value for all parameters that you don't want set to a default value. For more information, see [Updating user pool and app client configuration](#).

Include the following content in the LambdaVersion parameter of your request. A LambdaVersion value of V2_0 causes your user pool to add parameters for access token customization. To invoke a specific function version, use a Lambda function ARN with a function version as the value of LambdaArn.

```
"PreTokenGenerationConfig": {  
    "LambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:MyFunction",  
    "LambdaVersion": "V2_0"  
},
```

More resources

- [How to customize access tokens in Amazon Cognito user pools](#)

Topics

- [Pre token generation Lambda trigger sources](#)
- [Pre token generation Lambda trigger parameters](#)
- [Pre token trigger event version two example: Add and suppress claims, scopes, and groups](#)
- [Pre token generation event version two example: Add claims with complex objects](#)
- [Pre token generation event version one example: Add a new claim and suppress an existing claim](#)
- [Pre token generation event version one example: Modify the user's group membership](#)

Pre token generation Lambda trigger sources

triggerSource value	Event
TokenGeneration_HostedAuth	Called during authentication from the Amazon Cognito managed login sign-in page.
TokenGeneration_Authentication	Called after user authentication flows have completed.

triggerSource value	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

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests. When you add a pre token generation Lambda trigger to your user pool, you can choose a trigger version. This version determines whether Amazon Cognito passes a request to your Lambda function with additional parameters for access-token customization.

Version 1

The version 1 token can set group membership, IAM roles, and new claims in ID tokens.

```
{  
    "request": {  
        "userAttributes": {"string": "string"},  
        "groupConfiguration": {  
            "groupsToOverride": [  
                "string",  
                "string"  
            ],  
            "iamRolesToOverride": [  
                "string",  
                "string"  
            ],  
            "preferredRole": "string"  
        },  
        "clientMetadata": {"string": "string"}  
    },  
}
```

```
"response": {  
    "claimsOverrideDetails": {  
        "claimsToAddOrOverride": {"string": "string"},  
        "claimsToSuppress": [  
            "string",  
            "string"  
        ],  
        "groupOverrideDetails": {  
            "groupsToOverride": [  
                "string",  
                "string"  
            ],  
            "iamRolesToOverride": [  
                "string",  
                "string"  
            ],  
            "preferredRole": "string"  
        }  
    }  
}  
}
```

Version 2

The version 2 request event adds fields that customize the access token. It also adds support for complex claimsToOverride data types in the response object. Your Lambda function can return the following types of data in the value of claimsToOverride:

- String
- Number
- Boolean
- Array of strings, numbers, booleans, or a combination of any of these
- JSON

```
{  
    "request": {  
        "userAttributes": {  
            "string": "string"  
        },  
        "scopes": ["string", "string"],  
    }  
}
```

```

"groupConfiguration": {
    "groupsToOverride": ["string", "string"],
    "iamRolesToOverride": ["string", "string"],
    "preferredRole": "string"
},
"clientMetadata": {
    "string": "string"
}
},
"response": {
    "claimsAndScopeOverrideDetails": {
        "idTokenGeneration": {
            "claimsToAddOrOverride": {
                "string": [accepted datatype]
            },
            "claimsToSuppress": ["string", "string"]
        },
        "accessTokenGeneration": {
            "claimsToAddOrOverride": {
                "string": [accepted datatype]
            },
            "claimsToSuppress": ["string", "string"],
            "scopesToAdd": ["string", "string"],
            "scopesToSuppress": ["string", "string"]
        },
        "groupOverrideDetails": {
            "groupsToOverride": ["string", "string"],
            "iamRolesToOverride": ["string", "string"],
            "preferredRole": "string"
        }
    }
}
}

```

Pre token generation request parameters

Name	Description	Minimum trigger event version
userAttributes	The attributes of your user's profile in your user pool.	1

Name	Description	Minimum trigger event version
groupConfiguration	The input object that contains the current group configuration. The object includes groupsToOverride , iamRolesToOverride , and preferredRole .	1
groupsToOverride	The user pool groups that your user is a member of.	1
iamRolesToOverride	You can associate a user pool group with an AWS Identity and Access Management (IAM) role. This element is a list of all IAM roles from the groups that your user is a member of.	1
preferredRole	You can set a precedence for user pool groups. This element contains the name of the IAM role from the group with the highest precedence in the groupsToOverride element.	1
clientMetadata	One or more key-value pairs that you can specify and provide as custom input to the Lambda function for the pre token generation trigger. To pass this data to your Lambda function, use the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API operations. Amazon Cognito doesn't include data from the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations in the request that it passes to the pre token generation function.	1
scopes	Your user's OAuth 2.0 scopes. The scopes that are present in an access token are the user pool standard and custom scopes that your user requested, and that you authorized your app client to issue.	2

Pre token generation response parameters

Name	Description	Minimum trigger event version
claimsOverrideDetails	A container for all elements in a V1_0 trigger event.	1
claimsAndScopeOverrideDetails	A container for all elements in a V2_0 trigger event.	2
idTokenGeneration	The claims that you want to override, add, or suppress in your user's ID token. This parent to ID token customization values appears only in version 2 events, but the children elements appear in version 1 events.	2
accessTokenGeneration	The claims and scopes that you want to override, add, or suppress in your user's access token. This parent to access token customization values appears only in version 2 events.	2
claimsToAddOrOverride	A map of one or more claims and their values that you want to add or modify. For group-related claims, use groupOverrideDetails instead. In version 2 events, this element appears under both accessTokenGeneration and idTokenGeneration .	1*
claimsToSuppress	A list of claims that you want Amazon Cognito to suppress. If your function both suppresses and replaces a claim value, then Amazon Cognito suppresses the claim. In version 2 events, this element appears under both accessTokenGeneration and idTokenGeneration .	1

Name	Description	Minimum trigger event version
groupOverrideDetails	The output object that contains the current group configuration. The object includes groupsToOverride , iamRolesToOverride , and preferredRole . Your function replaces the groupOverrideDetails object with the object that you provide. If you provide an empty or null object in the response, then Amazon Cognito suppresses the groups. To keep the existing group configuration the same, copy the value of the groupConfiguration object of the request to the groupOverrideDetails object in the response. Then pass it back to the service. Amazon Cognito ID and access tokens both contain the cognito:groups claim. Your groupOverrideDetails object replaces the cognito:groups claim in access tokens and ID tokens.	1
scopesToAdd	A list of OAuth 2.0 scopes that you want to add to the scope claim in your user's access token. You can't add scope values that contain one or more blank-space characters.	2
scopesToDelete	A list of OAuth 2.0 scopes that you want to remove from the scope claim in your user's access token.	2

* Response objects to version 1 events can return strings. Response objects to version 2 events can return [complex objects](#).

Pre token trigger event version two example: Add and suppress claims, scopes, and groups

This example makes the following modifications to a user's tokens.

1. Sets their family_name as Doe in the ID token.
2. Prevents email and phone_number claims from appearing in the ID token.
3. Sets their ID token cognito:roles claim to "arn:aws:iam::123456789012:role\\sns_callerA", "arn:aws:iam::123456789012:role\\sns_callerC", "arn:aws:iam::123456789012:role\\sns_callerB".
4. Sets their ID token cognito:preferred_role claim to arn:aws:iam::123456789012:role/sns_caller.
5. Adds the scopes openid, email, and solar-system-data/asteroids.add to the access token.
6. Suppresses the scope phone_number and aws.cognito.signin.user.admin from the access token. Removal of phone_number prevents retrieval of the user's phone number from userInfo. Removal of aws.cognito.signin.user.admin prevents API requests by the user to read and modify their own profile with the Amazon Cognito user pools API.

 **Note**

The removal of phone_number from scopes only prevents retrieval of a user's phone number if the remaining scopes in the access token include openid and at least one more standard scope. For more information, see [About scopes](#).

7. Sets their ID and access token cognito:groups claim to "new-group-A", "new-group-B", "new-group-C".

JavaScript

```
export const handler = function(event, context) {  
    event.response = {  
        "claimsAndScopeOverrideDetails": {  
            "idTokenGeneration": {  
                "claimsToAddOrOverride": {  
                    "family_name": "Doe"  
                },  
                "claimsToSuppress": [  
                    "email",  
                    "phone_number"  
                ]  
            },  
            "accessTokenGeneration": {  
                "claimsToAddOrOverride": {  
                    "family_name": "Doe"  
                },  
                "scopes": ["openid", "email", "solar-system-data/asteroids.add"]  
            }  
        }  
    }  
}
```

```
        "scopesToAdd": [
            "openid",
            "email",
            "solar-system-data/asteroids.add"
        ],
        "scopesToSuppress": [
            "phone_number",
            "aws.cognito.signin.user.admin"
        ]
    },
    "groupOverrideDetails": {
        "groupsToOverride": [
            "new-group-A",
            "new-group-B",
            "new-group-C"
        ],
        "iamRolesToOverride": [
            "arn:aws:iam::123456789012:role/new_roleA",
            "arn:aws:iam::123456789012:role/new_roleB",
            "arn:aws:iam::123456789012:role/new_roleC"
        ],
        "preferredRole": "arn:aws:iam::123456789012:role/new_role",
    }
}
};

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

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

JSON

```
{
    "version": "2",
    "triggerSource": "TokenGeneration_Authentication",
    "region": "us-east-1",
    "userPoolId": "us-east-1_EXAMPLE",
    "userName": "JaneDoe",
```

```
"callerContext": {  
    "awsSdkVersion": "aws-sdk-unknown-unknown",  
    "clientId": "1example23456789"  
},  
"request": {  
    "userAttributes": {  
        "sub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",  
        "cognito:user_status": "CONFIRMED",  
        "email_verified": "true",  
        "phone_number_verified": "true",  
        "phone_number": "+12065551212",  
        "family_name": "Zoe",  
        "email": "Jane.Doe@example.com"  
    },  
    "groupConfiguration": {  
        "groupsToOverride": ["group-1", "group-2", "group-3"],  
        "iamRolesToOverride": ["arn:aws:iam::123456789012:role/sns_caller1",  
        "arn:aws:iam::123456789012:role/sns_caller2", "arn:aws:iam::123456789012:role/  
        sns_caller3"],  
        "preferredRole": ["arn:aws:iam::123456789012:role/sns_caller"]  
    },  
    "scopes": [  
        "aws.cognito.signin.user.admin", "openid", "email", "phone"  
    ]  
},  
"response": {  
    "claimsAndScopeOverrideDetails": []  
}  
}
```

Pre token generation event version two example: Add claims with complex objects

This example makes the following modifications to a user's tokens.

1. Adds claims of number, string, boolean, and JSON types to the ID token. This is the only change that version two trigger events makes available to the ID token.
2. Adds claims of number, string, boolean, and JSON types to the access token.
3. Adds three scopes to the access token.
4. Suppresses the email claim in the ID and access tokens.

5. Suppresses the aws.cognito.signin.user.admin scope in the access token.

JavaScript

```
export const handler = function(event, context) {

    var scopes = ["MyAPI.read", "MyAPI.write", "MyAPI.admin"]
    var claims = {}
    claims["aud"] = event.callerContext.clientId;
    claims["booleanTest"] = false;
    claims["longTest"] = 9223372036854775807;
    claims["exponentTest"] = 1.7976931348623157E308;
    claims["ArrayTest"] = ["test", 9223372036854775807, 1.7976931348623157E308,
true];
    claims["longStringTest"] = "\{\\"
        \"first_json_block\": \{\\"
            \"key_A\": \"value_A\", \\
            \"key_B\": \"value_B\"\\
        \}, \\
        \"second_json_block\": \{\\"
            \"key_C\": \{\\"
                \"subkey_D\": [\\
                    \"value_D\", \\
                    \"value_E\"\\
                ], \\
                \"subkey_F\": \"value_F\"\\
            \}, \\
            \"key_G\": \"value_G\"\\
        \}\\
    \}";
    claims["jsonTest"] = {
        "first_json_block": {
            "key_A": "value_A",
            "key_B": "value_B"
        },
        "second_json_block": {
            "key_C": {
                "subkey_D": [
                    "value_D",
                    "value_E"
                ],
                "subkey_F": "value_F"
            },
        }
    }
}
```

```
        "key_G": "value_G"
    }
};

event.response = {
    "claimsAndScopeOverrideDetails": {
        "idTokenGeneration": {
            "claimsToAddOrOverride": claims,
            "claimsToSuppress": ["email"]
        },
        "accessTokenGeneration": {
            "claimsToAddOrOverride": claims,
            "claimsToSuppress": ["email"],
            "scopesToAdd": scopes,
            "scopesToSuppress": ["aws.cognito.signin.user.admin"]
        }
    }
};

console.info("EVENT response\n" + JSON.stringify(event, (_, v) => typeof v === 'bigint' ? v.toString() : v, 2))
console.info("EVENT response size\n" + JSON.stringify(event, (_, v) => typeof v === 'bigint' ? v.toString() : v).length)
// Return to Amazon Cognito
context.done(null, event);
};
```

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

JSON

```
{
    "version": "2",
    "triggerSource": "TokenGeneration_HostedAuth",
    "region": "us-west-2",
    "userPoolId": "us-west-2_EXAMPLE",
    "userName": "JaneDoe",
    "callerContext": {
        "awsSdkVersion": "aws-sdk-unknown-unknown",
        "clientId": "1example23456789"
    },
}
```

```
"request": {  
    "userAttributes": {  
        "sub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",  
        "cognito:user_status": "CONFIRMED"  
        "email_verified": "true",  
        "phone_number_verified": "true",  
        "phone_number": "+12065551212",  
        "email": "Jane.Doe@example.com"  
    },  
    "groupConfiguration": {  
        "groupsToOverride": ["group-1", "group-2", "group-3"],  
        "iamRolesToOverride": ["arn:aws:iam::123456789012:role/sns_caller1"],  
        "preferredRole": ["arn:aws:iam::123456789012:role/sns_caller1"]  
    },  
    "scopes": [  
        "aws.cognito.signin.user.admin",  
        "phone",  
        "openid",  
        "profile",  
        "email"  
    ]  
},  
"response": {  
    "claimsAndScopeOverrideDetails": []  
}  
}
```

Pre token generation event version one example: Add a new claim and suppress an existing claim

This example uses a version 1 trigger event with a pre token generation Lambda function to add a new claim and suppresses an existing claim.

Node.js

```
const handler = async (event) => {  
  event.response = {  
    claimsOverrideDetails: {  
      claimsToAddOrOverride: {  
        my_first_attribute: "first_value",  
        my_second_attribute: "second_value",  
      },  
    },  
  }  
}
```

```
    claimsToSuppress: ["email"],  
  },  
};  
  
return event;  
};  
  
export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample: Because the code example doesn't process any request parameters, you can use a test event with an empty request. For more information about common request parameters, see [User pool Lambda trigger event](#).

JSON

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

Pre token generation event version one example: Modify the user's group membership

This example uses a version 1 trigger event with a pre token generation Lambda function to modify the user's group membership.

Node.js

```
const handler = async (event) => {  
  event.response = {  
    claimsOverrideDetails: {  
      groupOverrideDetails: {  
        groupsToOverride: ["group-A", "group-B", "group-C"],  
        iamRolesToOverride: [  
          "arn:aws:iam::XXXXXXXXXXXX:role/sns_callerA",  
          "arn:aws:iam::XXXXXXXXXXXX:role/sns_callerB",
```

```
        "arn:aws:iam::XXXXXXXXXX:role/sns_callerC",
    ],
    preferredRole: "arn:aws:iam::XXXXXXXXXX:role/sns_caller",
},
},
};

return event;
};

export { handler };
```

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

JSON

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

Migrate user Lambda trigger

When a user doesn't exist in the user pool at sign-in with a password, or in the forgot-password flow, Amazon Cognito invokes this trigger. 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 with a user migration Lambda trigger](#).

To migrate users from your existing user directory into Amazon Cognito user pools at sign-in, or during the forgot-password flow, use this Lambda trigger.

Topics

- [Migrate user Lambda trigger sources](#)
- [Migrate user Lambda trigger parameters](#)
- [Example: Migrate a user with an existing password](#)

Migrate user Lambda trigger sources

triggerSource value	Event
UserMigration_Authentication ¹	User migration at sign-in.
UserMigration_ForgotPassword	User migration during forgot-password flow.

¹ Amazon Cognito doesn't invoke this trigger when users authenticate with [passwordless sign-in](#).

Migrate user Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

JSON

```
{  
    "userAttributes": {  
        "string": "string",  
        . . .  
    },  
    "finalUserStatus": "string",  
    "messageAction": "string",  
    "desiredDeliveryMediums": [ "string", . . .],  
    "forceAliasCreation": boolean,  
    "enableSmsMfa": boolean  
}
```

```
}
```

Migrate user request parameters

userName

The username that the user enters at sign-in.

password

The password that the user enters at sign-in. Amazon Cognito doesn't send this value in a request that's initiated by a forgot-password flow.

validationData

One or more key-value pairs that contain the validation data in the user's sign-in request. To pass this data to your Lambda function, you can use 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 for the migrate user trigger. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the [AdminRespondToAuthChallenge](#) and [ForgotPassword](#) API actions.

Migrate user response parameters

userAttributes

This field is required.

This field must contain one or more name-value pairs that Amazon Cognito stores in the user profile in your user pool and uses as user attributes. 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

To reset their passwords in the forgot-password flow, a user must have either a verified email address or a verified phone number. Amazon Cognito sends a message containing a reset password code to the email address 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, Amazon Cognito uses default values.
username	<p>Required if you configured your user pool with alias attributes in addition to username for sign-in, and the user has entered a valid alias value as a username. This alias value can be an email address, preferred username, or phone number.</p> <p>If the request and the user pool meet the alias requirements, the response from your function must assign the <code>username</code> parameter that it received to an alias attribute. Also, the response must assign your own value to the <code>username</code> attribute. If your user pool doesn't meet the conditions required to map the received <code>username</code> to an alias, then the <code>username</code> parameter in the response must either exactly match the request, or be omitted.</p> <div data-bbox="584 1453 714 1493" data-label="Section-Header">Note</div> <div data-bbox="631 1507 1277 1548" data-label="Text"><p><code>username</code> must be unique in the user pool.</p></div>

finalUserStatus

You can set this parameter to CONFIRMED to auto-confirm your users so that they can sign in with their previous passwords. When you set a user to CONFIRMED, they do not need to take

additional action before they can sign in. If you don't set this attribute to CONFIRMED, it's set to RESET_REQUIRED.

A finalUserStatus of RESET_REQUIRED means that the user must change their password immediately after migration at sign-in, and your client app must handle the PasswordResetRequiredException during the authentication flow.

Note

Amazon Cognito doesn't enforce the password strength policy that you configured for the user pool during migration using Lambda trigger. If the password doesn't meet the password policy that you configured, Amazon Cognito still accepts the password so that it can continue to migrate the user. To enforce password strength policy and reject passwords that don't meet the policy, validate the password strength in your code.

Then, if the password doesn't meet the policy, set finalUserStatus to RESET_REQUIRED.

messageAction

You can set this parameter to SUPPRESS to decline to send the welcome message that Amazon Cognito usually sends to new users. If your function doesn't return this parameter, Amazon Cognito sends the welcome message.

desiredDeliveryMediums

You can set this parameter to EMAIL to send the welcome message by email, or SMS to send the welcome message by SMS. If your function doesn't return this parameter, Amazon Cognito sends the welcome message by SMS.

forceAliasCreation

If you set this parameter to TRUE and the phone number or email address in the UserAttributes parameter already exists as an alias with a different user, the API call migrates the alias from the previous user to the newly created user. The previous user can no longer log in using that alias.

If you set this parameter to FALSE and the alias exists, Amazon Cognito doesn't migrate the user and returns an error to the client app.

If you don't return this parameter, Amazon Cognito assumes its value is "false".

enableSMSMFA

Set this parameter to `true` to require that your migrated user complete SMS text message multi-factor authentication (MFA) to sign in. Your user pool must have MFA enabled. Your user's attributes in the request parameters must include a phone number, or else the migration of that user will fail.

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

```
const validUsers = {
  belladonna: { password: "Test123", emailAddress: "bella@example.com" },
};

// Replace this mock with a call to a real authentication service.
const authenticateUser = (username, password) => {
  if (validUsers[username] && validUsers[username].password === password) {
    return validUsers[username];
  }
  return null;
};

const lookupUser = (username) => {
  const user = validUsers[username];

  if (user) {
    return { emailAddress: user.emailAddress };
  }
  return null;
};

const handler = async (event) => {
  if (event.triggerSource === "UserMigration_Authentication") {
    // Authenticate the user with your existing user directory service
    const 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";
}
} else if (event.triggerSource === "UserMigration_ForgotPassword") {
// Look up the user in your existing user directory service
const 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";
}
}

return event;
};

export { handler };
```

Custom message Lambda trigger

When you have an external standard for the email and SMS messages that you want to send to your users, or when you want to apply your own logic at runtime to the formatting of user messages, add a custom message trigger to your user pool. The custom message Lambda receives the contents of all email and SMS messages before your user pool sends them. Your Lambda function then has the opportunity to modify the message contents and subject.

Amazon Cognito invokes this trigger before it sends an email or phone verification message or a multi-factor authentication (MFA) code. You can customize the message dynamically with your custom message trigger.

The request includes `codeParameter`. This is a string that acts as a placeholder for the code that Amazon Cognito delivers to the user. Insert the `codeParameter` string into the message body where you want the verification code to appear. When Amazon Cognito receives this response, Amazon Cognito replaces the `codeParameter` string with the actual verification code.

Note

A custom message Lambda function with the `CustomMessage_AdminCreateUser` trigger source returns a user name and verification code. Because an admin-created user must receive both their user name and code, the response from your function must include both `request.usernameParameter` and `request.codeParameter`.

Topics

- [Custom message Lambda trigger sources](#)
- [Custom message Lambda trigger parameters](#)
- [Custom message for sign-up example](#)
- [Custom message for admin create user example](#)

Custom message Lambda trigger sources

triggerSource value	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.
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

triggerSource value	Event
	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

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

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 user name. Amazon Cognito includes this parameter in requests that result from admin-created users.

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. The request that invokes a custom message function doesn't include data passed in the ClientMetadata parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the following API actions:

- [AdminResetUserPassword](#)
- [AdminRespondToAuthChallenge](#)
- [AdminUpdateUserAttributes](#)
- [ForgotPassword](#)
- [GetUserAttributeVerificationCode](#)
- [ResendConfirmationCode](#)
- [SignUp](#)
- [UpdateUserAttributes](#)

Custom message response parameters

In the response, specify the custom text to use in messages to your users. For the string constraints that Amazon Cognito applies to these parameters, see [MessageTemplateType](#).

smsMessage

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

emailMessage

The custom email message to send to your users. You can use HTML formatting in the emailMessage parameter. Must include the codeParameter value that you received in the request as the variable {####}. Amazon Cognito can use the emailMessage

parameter only if the `EmailSendingAccount` attribute of the user pool is `DEVELOPER`. If the `EmailSendingAccount` attribute of the user pool isn't `DEVELOPER` and an `emailMessage` parameter is returned, Amazon Cognito generates a 400 error code `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException`. When you choose Amazon Simple Email Service (Amazon SES) to send email messages, the `EmailSendingAccount` attribute of a user pool is `DEVELOPER`. Otherwise, the value is `COGNITO_DEFAULT`.

emailSubject

The subject line for the custom message. You can only use the `emailSubject` parameter if the `EmailSendingAccount` attribute of the user pool is `DEVELOPER`. If the `EmailSendingAccount` attribute of the user pool isn't `DEVELOPER` and Amazon Cognito returns an `emailSubject` parameter, Amazon Cognito generates a 400 error code `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException`. The `EmailSendingAccount` attribute of a user pool is `DEVELOPER` when you choose to use Amazon Simple Email Service (Amazon SES) to send email messages. Otherwise, the value is `COGNITO_DEFAULT`.

Custom message for sign-up example

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

Amazon Cognito can invoke a Lambda trigger at multiple events: post-registration, resending a verification code, recovering a forgotten password, or verifying a user attribute. The response includes messages for both SMS and email. The message must include the `code` parameter "####". This parameter is the placeholder for the verification code that the user receives.

The maximum length for an email message is 20,000 UTF-8 characters,. This length includes the verification code. You can use HTML tags in these email messages.

The maximum length of SMS messages is 140 UTF-8 characters. This length includes the verification code.

Node.js

```
const handler = async (event) => {
  if (event.triggerSource === "CustomMessage_SignUp") {
```

```
const message = `Thank you for signing up. Your confirmation code is
${event.request.codeParameter}.`;
event.response.smsMessage = message;
event.response.emailMessage = message;
event.response.emailSubject = "Welcome to the service.";
}
return event;
};

export { handler };
```

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

JSON

```
{
  "version": "1",
  "region": "us-west-2",
  "userPoolId": "us-west-2_EXAMPLE",
  "userName": "test-user",
  "callerContext": {
    "awsSdkVersion": "aws-sdk-unknown-unknown",
    "clientId": "1example23456789"
  },
  "triggerSource": "CustomMessage_SignUp",
  "request": {
    "userAttributes": {
      "sub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
      "cognito:user_status": "CONFIRMED",
      "email_verified": "true",
      "phone_number_verified": "true",
      "phone_number": "+12065551212",
      "email": "test-user@example.com"
    },
    "codeParameter": "{####}",
    "linkParameter": "{##Click Here##}",
    "usernameParameter": "None"
  },
  "response": {
```

```
    "smsMessage": "None",
    "emailMessage": "None",
    "emailSubject": "None"
}
}
```

Custom message for admin create user example

The request that Amazon Cognito sent to this example custom message Lambda function has a triggerSource value of `CustomMessage_AdminCreateUser` and a username and temporary password. The function populates `${event.request.codeParameter}` from the temporary password in the request, and `${event.request.usernameParameter}` from the username in the request.

Your custom messages must insert the values of `codeParameter` and `usernameParameter` into `smsMessage` and `emailMessage` in the response object. In this example, the function writes the same message to the response fields `event.response.smsMessage` and `event.response.emailMessage`.

The maximum length of an email message is 20,000 UTF-8 characters. This length includes the verification code. You can use HTML tags in these emails. The maximum length of SMS messages is 140 UTF-8 characters. This length includes the verification code.

The response includes messages for both SMS and email.

Node.js

```
const handler = async (event) => {
  if (event.triggerSource === "CustomMessage_AdminCreateUser") {
    const message = `Welcome to the service. Your user name is
    ${event.request.usernameParameter}. Your temporary password is
    ${event.request.codeParameter}`;
    event.response.smsMessage = message;
    event.response.emailMessage = message;
    event.response.emailSubject = "Welcome to the service";
  }
  return event;
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is 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

The Lambda triggers `CustomEmailSender` and `CustomSMSSender` support third-party email and SMS notifications in user pools. You can choose SMS and email providers to send notifications to users from within your Lambda function code. When Amazon Cognito sends invitations, MFA codes,

confirmation codes, verification codes, and temporary passwords to users, the events activate your configured Lambda functions. Amazon Cognito sends the code and temporary passwords (secrets) to your activated Lambda functions. Amazon Cognito encrypts these secrets with 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

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

CustomEmailSender

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

CustomSMSSender

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

Resources

Amazon Cognito doesn't send users' codes in plaintext in the events that it sends to custom sender triggers. The Lambda functions must decrypt codes in the events. The following concepts are the encryption architecture that your function must use to get codes that it can deliver to users.

AWS KMS

AWS KMS is a managed service to create and control AWS KMS keys. These keys encrypt your data. For more information see, [What is AWS Key Management Service?](#)

KMS key

A KMS key is a logical representation of a cryptographic key. The KMS key includes metadata, such as the key ID, creation date, description, and key state. The KMS key also contains the key material used to encrypt and decrypt data. For more information see, [AWS KMS keys](#).

Symmetric KMS key

A symmetric KMS key is a 256-bit encryption key that doesn't exit AWS KMS unencrypted. To use a symmetric KMS key, you must call AWS KMS. Amazon Cognito uses symmetric keys. The same key encrypts and decrypts. For more information see, [Symmetric KMS keys](#).

Custom email sender Lambda trigger

When you assign a custom email sender trigger to your user pool, Amazon Cognito invokes a Lambda function instead of its default behavior when a user event requires that it send an email message. With a custom sender trigger, your AWS Lambda function can send email notifications to your users through a method and provider that you choose. The custom code of your function must process and deliver all email messages from your user pool.

This trigger serves scenarios where you might want to have greater control over how your user pool sends email messages. Your Lambda function can customize the call to Amazon SES API operations, for example when you want to manage multiple verified identities or cross AWS Regions. Your function also might redirect messages to another delivery medium or third-party service.

Note

Currently, you can't assign custom sender triggers in the Amazon Cognito console.

You can assign a trigger with the `LambdaConfig` parameter in a `CreateUserPool` or `UpdateUserPool` API request.

To set up this trigger, perform the following steps:

1. Create a [symmetric encryption key](#) in AWS Key Management Service (AWS KMS). Amazon Cognito generates secrets—temporary passwords, verification codes, and confirmation codes—then uses this KMS key to encrypt the secrets. You can then use the [Decrypt](#) API operation in your Lambda function to decrypt the secrets and send them to the user in plaintext. The [AWS Encryption SDK](#) is a useful tool for AWS KMS operations in your function.
2. Create a Lambda function that you want to assign as your custom sender trigger. Grant `kms:Decrypt` permissions for your KMS key to the Lambda function role.
3. Grant Amazon Cognito service principal `cognito-idp.amazonaws.com` access to invoke the Lambda function.
4. Write Lambda function code that directs your messages to custom delivery methods or third-party providers. To deliver your user's verification or confirmation code, Base64 decode and decrypt the value of the `code` parameter in the request. This operation produces a plaintext code or password that you must include in your message.

5. Update the user pool so that it uses a custom sender Lambda trigger. The IAM principal that updates or creates a user pool with a custom sender trigger must have permission to create a grant for your KMS key. The following LambdaConfig snippet assigns custom SMS and email sender functions.

```
"LambdaConfig": {  
    "KMSKeyID": "arn:aws:kms:us-  
east-1:123456789012:key/a6c4f8e2-0c45-47db-925f-87854bc9e357",  
    "CustomEmailSender": {  
        "LambdaArn": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",  
        "LambdaVersion": "V1_0"  
    },  
    "CustomSMSSender": {  
        "LambdaArn": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",  
        "LambdaVersion": "V1_0"  
    }  
}
```

Custom email sender Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

JSON

```
{  
    "request": {  
        "type": "customEmailSenderRequestV1",  
        "code": "string",  
        "clientMetadata": {  
            "string": "string",  
            . . .  
        },  
        "userAttributes": {  
            "string": "string",  
            . . .  
        }  
    }  
}
```

Custom email sender request parameters

type

The request version. For a custom email sender event, the value of this string is always `customEmailSenderRequestV1`.

code

The encrypted code that your function can decrypt and send to your user.

clientMetadata

One or more key-value pairs that you can provide as custom input to the custom email sender Lambda function trigger. To pass this data to your Lambda function, you can use the `ClientMetadata` parameter in the [AdminRespondToAuthChallenge](#) and [RespondToAuthChallenge](#) API actions. Amazon Cognito doesn't include data from the `ClientMetadata` parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations in the request that it passes to the post authentication function.

Note

Amazon Cognito sends `ClientMetadata` to custom email trigger functions in events with the following trigger sources:

- `CustomEmailSender_ForgotPassword`
- `CustomEmailSender_SignUp`
- `CustomEmailSender_Authentication`

Amazon Cognito doesn't send `ClientMetadata` in trigger events with source `CustomEmailSender_AccountTakeOverNotification`.

userAttributes

One or more key-value pairs that represent user attributes.

Custom email sender response parameters

Amazon Cognito doesn't expect any additional return information in the custom email sender response. Your Lambda function must interpret the event and decrypt the code, then deliver the

message contents. A typical function assembles an email message and directs it to a third-party SMTP relay.

Activating the custom email sender Lambda trigger

To set up a custom email sender trigger that uses custom logic to send email messages for your user pool, activate the trigger as follows. The procedure that follows assigns a custom email trigger, a custom SMS trigger, or both to your user pool. After you add your custom email sender trigger, Amazon Cognito always sends user attributes, including the email address, and the one-time code to your Lambda function when it would have otherwise sent an email message with Amazon Simple Email Service.

Important

Amazon Cognito HTML-escapes reserved characters like < (<) and > (>) in your user's temporary password. These characters might appear in temporary passwords that Amazon Cognito sends to your custom email sender function, but don't appear in temporary verification codes. To send temporary passwords, your Lambda function must unescape these characters after it decrypts the password, and before it sends the message to your user.

1. Create an encryption key in AWS KMS. This key encrypts temporary passwords and authorization codes that Amazon Cognito generates. You can then decrypt these secrets in the custom sender Lambda function and send them to your user in plaintext.
2. Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to encrypt codes with the KMS key.

Apply the following resource-based policy to your KMS key.

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "cognito-idp.amazonaws.com"  
        },  
        "Action": "kms>CreateGrant",  
        "Resource": "arn:aws:kms:us-west-2:111222333444:key/1example-2222-3333-4444-999example",  
        "Condition": {}  
    }]  
}
```

```
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "111222333444"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:cognito-idp:us-west-2:111222333444:userpool/us-east-1_EXAMPLE"
            }
        }
    }
}
```

3. Create a Lambda function for the custom sender trigger. Amazon Cognito uses the [AWS encryption SDK](#) to encrypt the secrets, temporary passwords and codes that authorize your users' API requests.
 - Assign an IAM role to your Lambda function that has, at minimum, kms:Decrypt permissions for your KMS key.
4. Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function.

The following AWS CLI command grants Amazon Cognito permission to invoke your Lambda function:

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

5. Compose your Lambda function code to send your messages. Amazon Cognito uses AWS Encryption SDK to encrypt secrets before Amazon Cognito sends the secrets to the custom sender Lambda function. In your function, decrypt the secret and process any relevant metadata. Then send the code, your own custom message, and destination phone number to the custom API that delivers your message.
6. Add the AWS Encryption SDK to your Lambda function. For more information, see [AWS Encryption SDK programming languages](#). To update the Lambda package, complete the following steps.
 - a. Export your Lambda function as a .zip file in the AWS Management Console.

- b. Open your function and add the AWS Encryption SDK. For more information and download links, see [AWS Encryption SDK programming languages](#) in the *AWS Encryption SDK Developer Guide*.
 - c. Zip your function with your SDK dependencies, and upload the function to Lambda. For more information, see [Deploying Lambda functions as .zip file archives](#) in the *AWS Lambda Developer Guide*.
7. Update your user pool to add custom sender Lambda triggers. Include a `CustomSMSender` or `CustomEmailSender` parameter in an `UpdateUserPool` API request. The `UpdateUserPool` API operation requires all the parameters of your user pool *and* the parameters that you want to change. If you don't provide all relevant parameters, Amazon Cognito sets the values of any missing parameters to their defaults. As demonstrated in the example that follows, include entries for all Lambda functions that you want to add to or keep in your user pool. For more information, see [Updating user pool and app client configuration](#).

```
#Send this parameter in an 'aws cognito-identity update-user-pool' CLI command,  
including any existing  
#user pool configurations.  
  
--lambda-config "PreSignUp=Lambda-arn, \  
                  CustomSMSender=[LambdaVersion=V1_0,LambdaArn=Lambda-arn], \  
                  CustomEmailSender=[LambdaVersion=V1_0,LambdaArn=Lambda-arn],  
                  \  
                  KMSKeyID=key-id"
```

To remove a custom sender Lambda trigger with an `update-user-pool` AWS CLI, omit the `CustomSMSender` or `CustomEmailSender` parameter from `--lambda-config`, and include all other triggers that you want to use with your user pool.

To remove a custom sender Lambda trigger with an `UpdateUserPool` API request, omit the `CustomSMSender` or `CustomEmailSender` parameter from the request body that contains the rest of your user pool configuration.

Code example

The following Node.js example processes an email message event in your custom email sender Lambda function. This example assumes your function has two environment variables defined.

KEY_ALIAS

The [alias](#) of the KMS key that you want to use to encrypt and decrypt your users' codes.

KEY_ARN

The Amazon Resource Name (ARN) of the KMS key that you want to use to encrypt and decrypt your users' codes.

```
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_ARN ];
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 contains the decrypted secret.
  if(event.triggerSource == 'CustomEmailSender_SignUp'){
    //Send an email message to your user via a custom provider.
    //Include the temporary password in the message.
  }
  else if(event.triggerSource == 'CustomEmailSender_Authentication'){
    //Send an MFA message.
  }
  else if(event.triggerSource == 'CustomEmailSender_Resendcode'){
    //Send a message with next steps for password reset.
  }
  else if(event.triggerSource == 'CustomEmailSender_ForgotPassword'){
    //Send a message with next steps for password reset.
  }
  else if(event.triggerSource == 'CustomEmailSender_UpdateUserAttribute'){
    //Send a message with next steps for confirming the new attribute.
  }
}
```

```
else if(event.triggerSource == 'CustomEmailSender_VerifyUserAttribute'){
    //Send a message with next steps for confirming the new attribute.
}
else if(event.triggerSource == 'CustomEmailSender_AdminCreateUser'){
    //Send a message with next steps for signing in with a new user profile.
}
else if(event.triggerSource == 'CustomEmailSender_AccountTakeOverNotification'){
    //Send a message describing the threat protection event and next steps.
}
return;
};
```

Custom email sender Lambda trigger sources

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

TriggerSource value	Event
CustomEmailSender_SignUp	A user signs up and Amazon Cognito sends a welcome message.
CustomEmailSender_Authentication	A user signs in and Amazon Cognito sends an multi-factor authentication (MFA) code.
CustomEmailSender_ForgotPassword	A user requests a code to reset their password.
CustomEmailSender_Resendcode	A user requests a replacement account-confirmation code.
CustomEmailSender_UpdateUserAttribute	A user updates an email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomEmailSender_VerifyUserAttribute	A user creates a new email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.

TriggerSource value	Event
CustomEmailSender_AdminCreateUser	You create a new user in your user pool and Amazon Cognito sends them a temporary password.
CustomEmailSender_AccountTakeOverNotification	Amazon Cognito detects an attempt to take over a user account and sends the user a notification.

Custom SMS sender Lambda trigger

When you assign a custom SMS sender trigger to your user pool, Amazon Cognito invokes a Lambda function instead of its default behavior when a user event requires that it send an SMS message. With a custom sender trigger, your AWS Lambda function can send SMS notifications to your users through a method and provider that you choose. The custom code of your function must process and deliver all SMS messages from your user pool.

This trigger serves scenarios where you might want to have greater control over how your user pool sends SMS messages. Your Lambda function can customize the call to Amazon SNS API operations, for example when you want to manage multiple origination IDs or cross AWS Regions. Your function also might redirect messages to another delivery medium or third-party service.

 **Note**

Currently, you can't assign custom sender triggers in the Amazon Cognito console.

You can assign a trigger with the `LambdaConfig` parameter in a `CreateUserPool` or `UpdateUserPool` API request.

To set up this trigger, perform the following steps:

1. Create a [symmetric encryption key](#) in AWS Key Management Service (AWS KMS). Amazon Cognito generates secrets—temporary passwords, verification codes, and confirmation codes—then uses this KMS key to encrypt the secrets. You can then use the [Decrypt](#) API operation in your Lambda function to decrypt the secrets and send them to the user in plaintext. The [AWS Encryption SDK](#) is a useful tool for AWS KMS operations in your function.

2. Create a Lambda function that you want to assign as your custom sender trigger. Grant kms:Decrypt permissions for your KMS key to the Lambda function role.
3. Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function.
4. Write Lambda function code that directs your messages to custom delivery methods or third-party providers. To deliver your user's verification or confirmation code, Base64 decode and decrypt the value of the code parameter in the request. This operation produces a plaintext code or password that you must include in your message.
5. Update the user pool so that it uses a custom sender Lambda trigger. The IAM principal that updates or creates a user pool with a custom sender trigger must have permission to create a grant for your KMS key. The following LambdaConfig snippet assigns custom SMS and email sender functions.

```
"LambdaConfig": {  
    "KMSKeyID": "arn:aws:kms:us-east-1:123456789012:key/a6c4f8e2-0c45-47db-925f-87854bc9e357",  
    "CustomEmailSender": {  
        "LambdaArn": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",  
        "LambdaVersion": "V1_0"  
    },  
    "CustomSMSSender": {  
        "LambdaArn": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",  
        "LambdaVersion": "V1_0"  
    }  
}
```

Custom SMS sender Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the [common parameters](#) that Amazon Cognito adds to all requests.

JSON

```
{  
    "request": {  
        "type": "customSMSSenderRequestV1",  
        "code": "string",  
        "clientMetadata": {  
            "string": "string",  
            . . .  
        }  
    }  
}
```

```
        },
        "userAttributes": {
            "stringstring",
            ...
        }
    }
```

Custom SMS sender request parameters

type

The request version. For a custom SMS sender event, the value of this string is always `customSMSenderRequestV1`.

code

The encrypted code that your function can decrypt and send to your user.

clientMetadata

One or more key-value pairs that you can provide as custom input to the custom SMS sender Lambda function trigger. To pass this data to your Lambda function, you can use the `ClientMetadata` parameter in the [AdminRespondToAuthChallenge](#) and [RespondToAuthChallenge](#) API actions. Amazon Cognito doesn't include data from the `ClientMetadata` parameter in [AdminInitiateAuth](#) and [InitiateAuth](#) API operations in the request that it passes to the post authentication function.

userAttributes

One or more key-value pairs that represent user attributes.

Custom SMS sender response parameters

Amazon Cognito doesn't expect any additional return information in the response. Your function can use API operations to query and modify your resources, or record event metadata to an external system.

Activating the custom SMS sender Lambda trigger

You can set up a custom SMS sender trigger that uses custom logic to send SMS messages for your user pool. The following procedure assigns a custom SMS trigger, a custom email trigger, or both to

your user pool. After you add your custom SMS sender trigger, Amazon Cognito always sends user attributes, including the phone number, and the one-time code to your Lambda function instead of the default behavior that sends an SMS message with Amazon Simple Notification Service.

Important

Amazon Cognito HTML-escapes reserved characters like < (<) and > (>) in your user's temporary password. These characters might appear in temporary passwords that Amazon Cognito sends to your custom email sender function, but don't appear in temporary verification codes. To send temporary passwords, your Lambda function must unescape these characters after it decrypts the password, and before it sends the message to your user.

1. Create an encryption key in AWS KMS. This key encrypts temporary passwords and authorization codes that Amazon Cognito generates. You can then decrypt these secrets in the custom sender Lambda function and send them to your user in plaintext.
2. Grant Amazon Cognito service principal `cognito-idp.amazonaws.com` access to encrypt codes with the KMS key.

Apply the following resource-based policy to your KMS key.

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "cognito-idp.amazonaws.com"  
        },  
        "Action": "kms>CreateGrant",  
        "Resource": "arn:aws:kms:us-west-2:111222333444:key/1example-2222-3333-4444-999example",  
        "Condition": {  
            "StringEquals": {  
                "aws:SourceAccount": "111222333444"  
            },  
            "ArnLike": {  
                "aws:SourceArn": "arn:aws:cognito-idp:us-west-2:111222333444:userpool/us-east-1_EXAMPLE"  
            }  
        }  
    }]
```

```
        }  
    ]]  
}
```

3. Create a Lambda function for the custom sender trigger. Amazon Cognito uses the [AWS encryption SDK](#) to encrypt the secrets, temporary passwords and codes that authorize your users' API requests.
 - Assign an IAM role to your Lambda function that has, at minimum, kms:Decrypt permissions for your KMS key.
4. Grant Amazon Cognito service principal cognito-idp.amazonaws.com access to invoke the Lambda function.

The following AWS CLI command grants Amazon Cognito permission to invoke your Lambda function:

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

5. Compose your Lambda function code to send your messages. Amazon Cognito uses AWS Encryption SDK to encrypt secrets before Amazon Cognito sends the secrets to the custom sender Lambda function. In your function, decrypt the secret and process any relevant metadata. Then send the code, your own custom message, and destination phone number to the custom API that delivers your message.
6. Add the AWS Encryption SDK to your Lambda function. For more information, see [AWS Encryption SDK programming languages](#). To update the Lambda package, complete the following steps.
 - a. Export your Lambda function as a .zip file in the AWS Management Console.
 - b. Open your function and add the AWS Encryption SDK. For more information and download links, see [AWS Encryption SDK programming languages](#) in the *AWS Encryption SDK Developer Guide*.
 - c. Zip your function with your SDK dependencies, and upload the function to Lambda. For more information, see [Deploying Lambda functions as .zip file archives](#) in the *AWS Lambda Developer Guide*.

7. Update your user pool to add custom sender Lambda triggers. Include a `CustomSMSSender` or `CustomEmailSender` parameter in an `UpdateUserPool` API request. The `UpdateUserPool` API operation requires all the parameters of your user pool *and* the parameters that you want to change. If you don't provide all relevant parameters, Amazon Cognito sets the values of any missing parameters to their defaults. As demonstrated in the example that follows, include entries for all Lambda functions that you want to add to or keep in your user pool. For more information, see [Updating user pool and app client configuration](#).

```
#Send this parameter in an 'aws cognito-identity update-user-pool' CLI command,  
including any existing  
#user pool configurations.  
  
--lambda-config "PreSignUp=lambda-arn, \  
                  CustomSMSSender={LambdaVersion=V1_0,LambdaArn=lambda-arn}, \  
                  CustomEmailSender={LambdaVersion=V1_0,LambdaArn=lambda-arn}, \  
                  \  
                  KMSKeyID=key-id"
```

To remove a custom sender Lambda trigger with an `update-user-pool` AWS CLI, omit the `CustomSMSSender` or `CustomEmailSender` parameter from `--lambda-config`, and include all other triggers that you want to use with your user pool.

To remove a custom sender Lambda trigger with an `UpdateUserPool` API request, omit the `CustomSMSSender` or `CustomEmailSender` parameter from the request body that contains the rest of your user pool configuration.

Code example

The following Node.js example processes an SMS message event in your custom SMS sender Lambda function. This example assumes your function has two environment variables defined.

KEY_ALIAS

The [alias](#) of the KMS key that you want to use to encrypt and decrypt your users' codes.

KEY_ARN

The Amazon Resource Name (ARN) of the KMS key that you want to use to encrypt and decrypt your users' codes.

```
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_ARN ];
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 contains the decrypted secret.
  if(event.triggerSource == 'CustomSMSSender_SignUp'){
    //Send an SMS message to your user via a custom provider.
    //Include the temporary password in the message.
  }
  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

- [Evaluate SMS message capabilities with a custom SMS sender function](#)
- [Custom SMS sender Lambda trigger sources](#)

Evaluate SMS message capabilities with a custom SMS sender function

A custom SMS sender Lambda function accepts the SMS messages that your user pool would send, and the function delivers the content based on your custom logic. Amazon Cognito sends the [Custom SMS sender Lambda trigger parameters](#) to your function. Your function can do what you want with this information. For example, you can send the code to an Amazon Simple Notification Service (Amazon SNS) topic. An Amazon SNS topic subscriber can be an SMS message, an HTTPS endpoint, or an email address.

To create a test environment for Amazon Cognito SMS messaging with a custom SMS sender Lambda function, see [amazon-cognito-user-pool-development-and-testing-with-sms-redirected-to-email](#) in the [aws-samples library on GitHub](#). The repository contains AWS CloudFormation templates that can create a new user pool, or work with a user pool that you already have. These templates create Lambda functions and an Amazon SNS topic. The Lambda function that the template assigns as a custom SMS sender trigger, redirects your SMS messages to the subscribers to the Amazon SNS topic.

When you deploy this solution to a user pool, all messages that Amazon Cognito usually sends through SMS messaging, the Lambda function instead sends to a central email address. Use this solution to customize and preview SMS messages, and to test the user pool events that cause Amazon Cognito to send an SMS message. After you complete your tests, roll back the CloudFormation stack, or remove the custom SMS sender function assignment from your user pool.

Important

Don't use the templates in [amazon-cognito-user-pool-development-and-testing-with-sms-redirected-to-email](#) to build a production environment. The custom SMS sender Lambda function in the solution *simulates* SMS messages, but the Lambda function sends them all to a single central email address. Before you can send SMS messages in a production Amazon Cognito user pool, you must complete the requirements shown at [SMS message settings for Amazon Cognito user pools](#).

Custom SMS sender Lambda trigger sources

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

TriggerSource value	Event
CustomSMSender_SignUp	A user signs up and Amazon Cognito sends a welcome message.
CustomSMSender_ForgotPassword	A user requests a code to reset their password.
CustomSMSender_Resendcode	A user requests a new code to confirm their registration.
CustomSMSender_VerifyUserAttribute	A user creates a new email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomSMSender_UpdateUserAttribute	A user updates an email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomSMSender_Authentication	A user configured with SMS multi-factor authentication (MFA) signs in.
CustomSMSender_AdminCreateUser	You create a new user in your user pool and Amazon Cognito sends them a temporary password.

Managing users in your user pool

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

Managing users in your Amazon Cognito user pool involves a variety of configuration options and administrative tasks. User pools can scale to millions of users. A user directory of this scale requires equally scalable and repeatable administrative tools. You might want to create many user profiles, manage inactive users, produce governance and compliance reports, or set up self-service tools where users do most of the work. After you create a user pool, you can control how users sign up and confirm their accounts, including requiring email or phone number verification.

Administrators can also create user accounts directly and customize the welcome messages and password requirements.

User pools have user groups, where you can manage access to resources based on a user's group membership. You can assign IAM roles to these groups to manage access to AWS services with identity pools. Users' group membership is present in both ID and access tokens. With this information, you can make access-control decisions at runtime in your application or with a policy engine like Amazon Verified Permissions.

User pools often have many users. You will frequently find yourself searching for and updating user accounts. The Amazon Cognito console and API support querying users based on standard attributes like username, email, and phone number. Administrators can also reset passwords, disable accounts, and view user event history.

For migrating existing user data, Amazon Cognito has options to import users from a CSV file and to use a [Lambda trigger](#) to automatically migrate users when they first sign in. These options support user transitions from other user directories to your user pool.

You can use the user-management features in user pools to have fine-grained control over the user lifecycle and authentication experience. The combination of self-service sign-up, admin-created accounts, groups, and migration tools makes Amazon Cognito user pools a flexible user directory.

Topics

- [Configuring policies for user creation](#)
- [Signing up and confirming user accounts](#)
- [Creating user accounts as administrator](#)
- [Adding groups to a user pool](#)
- [Managing and searching for user accounts](#)
- [Passwords, account recovery, and password policies](#)
- [Importing users into a user pool](#)
- [Working with user attributes](#)

Configuring policies for user creation

Your user pool can allow users to sign up, or you can create them as an administrator. You can also control how much of the process of verification and confirmation after sign-up is in the hands of

your users. For example, you might want to review sign-ups and accept them based on an external validation process. This configuration, or *admin create user policy*, also sets the amount of time before a user can no longer confirm their user account.

Amazon Cognito can serve the needs of your public customers as the customer identity and access management (CIAM) platform for your software. A user pool that accepts sign-up and has an app client, with or without managed login, creates a user profile for anyone on the internet who knows your publicly-discoverable app client ID and requests to sign up. A signed-up user profile can receive access and identity tokens and can access resources that you've authorized for your app. Before you activate sign-up in your user pool, review your options and ensure that your configuration complies with your security standards. Set **Enable self-registration** and **AllowAdminCreateUserOnly**, described in the following procedures, with care.

AWS Management Console

The **Sign-up** menu of your user pool contains some of the settings for sign-up and administrative creation of users in your user pool.

To configure the sign-up experience

1. In **Cognito-assisted verification and confirmation**, choose whether you want to **Allow Cognito to automatically send messages to verify and confirm**. With this setting enabled, Amazon Cognito sends an email or SMS message to new users with a code that they must present to your user pool. This confirms their ownership of the email address or phone number, setting the equivalent attribute as verified and confirming the user account for sign-in. The **Attributes to verify** that you choose determine the delivery methods and destinations of the verification messages.
2. **Verifying attribute changes** isn't significant when you're creating users, but relates to attribute verification. You can permit users who have changed but not yet verified their [sign-in attributes](#) to continue to sign in either with their new attribute value or with their original. For more information, see [Verifying when users change their email or phone number](#).
3. **Required attributes** displays the attributes that must be provided a value before a user can sign up or you can create a user. You can only set required attributes in the create user pool wizard.
4. **Custom attributes** are important to the user creation and sign-up process because you can only set a value for *immutable* custom attributes when you first create a user. For more information about custom attributes, see [Custom attributes](#).

5. In **Self-service sign-up**, select **Enable self-registration** if you want users to be able to generate a new account with the [unauthenticated](#) SignUp API. If you disable self-registration, you can only create new users as an administrator, in the Amazon Cognito console or with [AdminCreateUser](#) API requests. In a user pool where self-registration is inactive, [SignUp](#) API requests return NotAuthorizedException and managed login doesn't display a **Sign up** link.

For user pools where you plan to create users as an administrator, you can configure the duration of their temporary passwords in the setting in the **Sign-in** menu **Temporary passwords set by administrators expire in**.

Another important element of the creation of users as an administrator is the invitation message. When you create a new user, Amazon Cognito sends them a message with a link to your app so that they can sign in for the first time. Customize this message template in the **Authentication methods** menu under **Message templates**.

You can configure [confidential app clients](#), typically web applications, with a client secret that prevents sign-up without the app client secret. As a security best practice, do not distribute app client secrets in public app clients, typically mobile apps. You can create app clients with client secrets in the **App clients** menu of the Amazon Cognito console.

Amazon Cognito user pools API

You can programmatically set the parameters for creation of users in a user pool in a [CreateUserPool](#) or [UpdateUserPool](#) API request.

The [AdminCreateUserConfig](#) element sets values for the following properties of a user pool.

1. Enable self-service sign-up
2. The invitation message that you send to new admin-created users

The following example, when added to a full API request body, sets a user pool with self-service sign-up inactive and a basic invitation email.

```
"AdminCreateUserConfig": {  
    "AllowAdminCreateUserOnly": true,  
    "InviteMessageTemplate": {  
        "EmailMessage": "Your username is {username} and temporary password is  
        {####}.",  
    },  
},
```

```
        "EmailSubject": "Welcome to ExampleApp",
        "SMSMessage": "Your username is {username} and temporary password is
{####}."
    }
}
```

The following additional parameters of a [CreateUserPool](#) or [UpdateUserPool](#) API request govern the creation of new users.

AutoVerifiedAttributes

The attributes, email addresses or phone numbers, that you want to [automatically send a message to](#) when you register a new user.

Policies

The user pool [password policy](#).

Schema

The user pool [custom attributes](#). They are important to the user creation and sign-up process because you can only set a value for *immutable* custom attributes when you first create a user.

This parameter also sets the required attributes for your user pool. The following text, when inserted into the Schema element of a full API request body, set the email attribute as required.

```
{
    "Name": "email",
    "Required": true
}
```

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. This 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](#).

- 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](#).

Users who sign themselves up must be confirmed before they can sign in. Imported and created users are already confirmed, but they must create their password the first time they sign in. The following sections explain the confirmation process and email and phone verification.

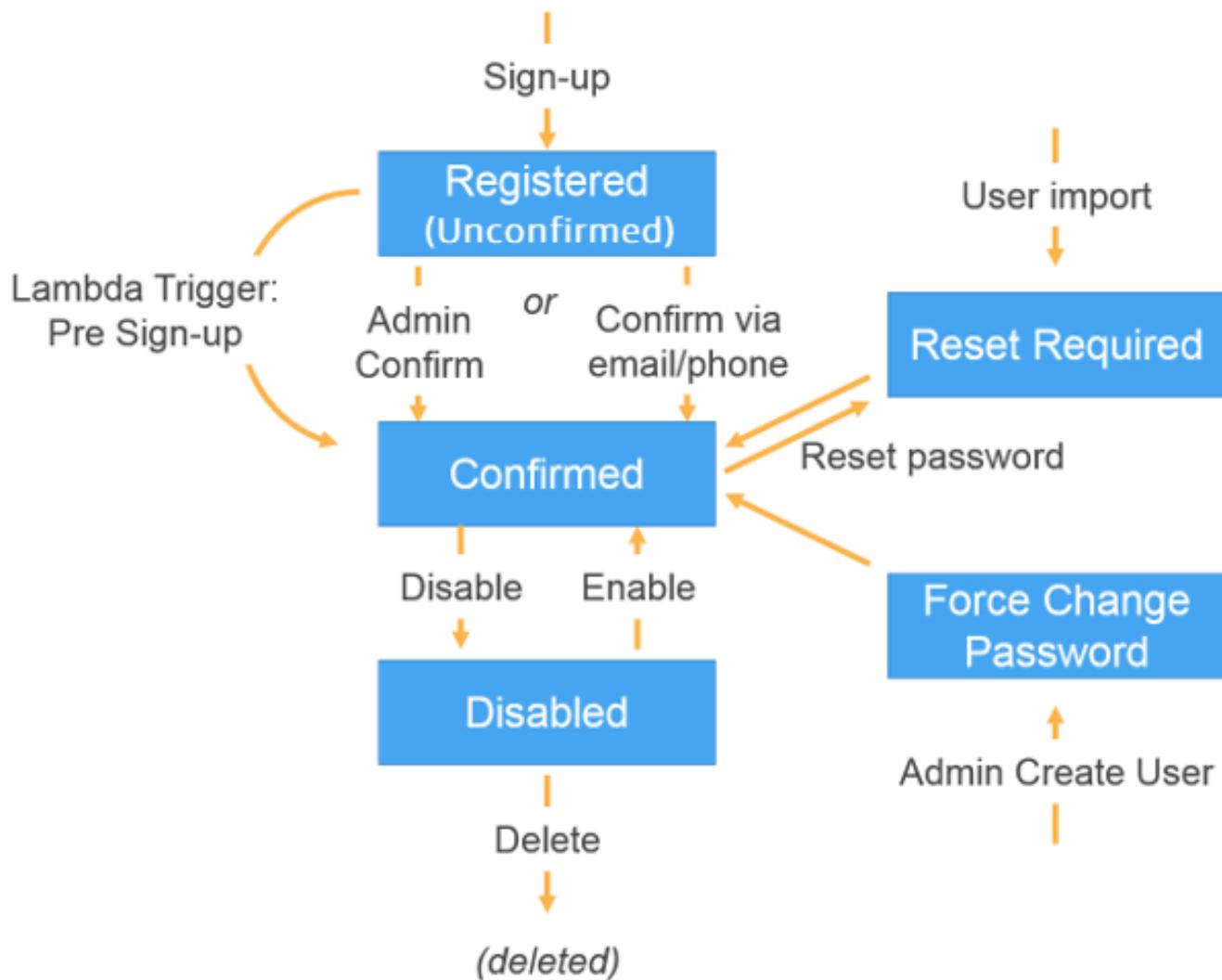
Passwords at sign-up

Amazon Cognito requires passwords from all users when they sign up, except under the following conditions. If *all* of these conditions are met, you can omit passwords in sign-up operations.

1. [Passwordless sign-in](#) is active in your user pool and app client.
2. Your application is custom-built with authentication modules in an AWS SDK. Managed login and the hosted UI always require passwords.
3. Users provide attribute values for the passwordless sign-in methods—email or SMS message one-time passwords (OTPs)—that you permit. For example, if you allow sign-in with email and phone OTP, users can provide either a phone number or email address, but if you only allow sign-in with email, they must provide an email address.
4. Your user pool [automatically verifies](#) the attributes that users can use with passwordless sign-in.
5. For any given [SignUp](#) request, the user doesn't provide a value for the [Password](#) parameter.

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. When a user enters a code or follows an email link to confirm their user account, 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 their password before they 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 their 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 you can delete a user account, you must disable sign-in access for that user.

More resources

- [Detecting and remediating inactive user accounts with Amazon Cognito](#)

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 their 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. If 24 hours have passed and your user's code or link is no longer valid, call the [ResendConfirmationCode](#) API operation to generate and send a new code or link.

By providing the verification code, a user proves that they have 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 `Admin GetUser` API operation, 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.

Other actions that confirm and verify user attributes

The following user activity verifies user attributes. You're not required to set these attributes to automatically verify: the listed actions mark them as verified in all cases.

Email address

1. Successfully completing [passwordless authentication](#) with an email one-time password (OTP).
2. Successfully completing [multi-factor authentication \(MFA\)](#) with an email OTP.

Phone number

1. Successfully completing [passwordless authentication](#) with an SMS OTP.
2. Successfully completing [MFA](#) with an SMS OTP.

To configure your user pool to require email or phone verification

When you verify your users' email addresses and phone numbers, you ensure that you can contact your users. Complete the following steps in the AWS Management Console to configure your user pool to require that your users confirm their email addresses or phone numbers.

Note

If you don't yet have a user pool in your account, see [Getting started with user pools](#).

To configure your user pool

1. Navigate to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. From the navigation pane, choose **User Pools**. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Sign-up** menu and locate **Attribute verification and user account confirmation**. Choose **Edit**.
4. Under **Cognito-assisted verification and confirmation**, choose whether you will **Allow Cognito to automatically send messages to verify and confirm**. With this setting enabled, Amazon Cognito sends messages to the user contact attributes you choose when a user signs up, or you create a user profile. To verify attributes and confirm user profiles for sign-in, Amazon Cognito sends a code or link in messages to users. The users must then enter the code in your UI so that your app can confirm them in a `ConfirmSignUp` or `AdminConfirmSignUp` API request.

Note

You can also disable **Cognito-assisted verification and confirmation** and use authenticated API actions or Lambda triggers to verify attributes and confirm users. 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 may be unable to use your app. Remember that users require verified contact information to:

- **Reset their passwords** — When a user chooses an option 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 [Customizing sign-in attributes](#).

5. Choose your **Attributes to verify**:

Send SMS message, verify phone number

Amazon Cognito sends a verification code in an SMS message when the user signs up. Choose this option if you typically communicate with your users through SMS messages. For example, you will want to use verified phone numbers if you send delivery notifications, appointment confirmations, or alerts. User phone numbers will be the verified attribute when accounts are confirmed; you must take additional action to verify and communicate with user email addresses.

Send email message, verify email address

Amazon Cognito sends a verification code through an email message 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. User email addresses will be the verified attribute when accounts are confirmed; you must take additional action to verify and communicate with user phone numbers.

Send SMS message if phone number is available, otherwise send email message

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](#).

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 (in an SMS message) 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 address 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 message. 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 their 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 a user pool administrator

You might not want your user pool to automatically send verification messages in your user pool, but still want to allow anyone to sign up for an account. This model leaves room, for example, for human review of new sign-up requests, and for batch validation and processing of sign-ups. You can confirm new user accounts in the Amazon Cognito console or with the IAM-authenticated API operation [AdminConfirmSignUp](#). You can confirm user accounts as an administrator whether or not your user pool sends verification messages.

You can only confirm a user self-service sign-up with this technique. To confirm a user that you create as an administrator, create an [AdminSetUserPassword](#) API request with Permanent set to True.

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. Confirm the user's account. You must sign in to the AWS Management Console or sign your API request with AWS credentials to confirm the account.
 - a. To confirm a user in the Amazon Cognito console, navigate to the **Users** menu, choose the user who you want to confirm, and from the **Actions** menu select **Confirm**.
 - b. To confirm a user in the AWS API or CLI, create a [AdminConfirmSignUp](#) API request, or [admin-confirm-sign-up](#) in the AWS CLI.
4. At this point the user's account is in a confirmed state, and the user can sign in.

Computing secret hash values

Assign a client secret to your confidential app client as a best practice. When you assign a client secret to your app client, your Amazon Cognito user pools API requests must include a hash that

includes the client secret in the request body. To validate your knowledge of the client secret for the API operations in the following lists, concatenate the client secret with your app client ID and your user's username, then base64-encode that string.

When your app signs in users to a client that has a secret hash, you can use the value of any user pool sign-in attribute as the username element of the secret hash. When your app requests new tokens in an authentication operation with REFRESH_TOKEN_AUTH, the value of the username element depends on your sign-in attributes. When your user pool doesn't have username as a sign-in attribute, set the secret hash username value from the user's sub claim from their access or ID token. When username is a sign-in attribute, set the secret hash username value from the username claim.

The following Amazon Cognito user pools APIs accept a client-secret hash value in a SecretHash parameter.

- [ConfirmForgotPassword](#)
- [ConfirmSignUp](#)
- [ForgotPassword](#)
- [ResendConfirmationCode](#)
- [SignUp](#)

Additionally, the following APIs accept a client-secret hash value in a SECRET_HASH parameter, either in authentication parameters or in a challenge response.

API operation	Parent parameter for SECRET_HASH
InitiateAuth	AuthParameters
AdminInitiateAuth	AuthParameters
RespondToAuthChallenge	ChallengeResponses
AdminRespondToAuthChallenge	ChallengeResponses

The secret hash 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" ) )
```

For a detailed overview of how to calculate and use the SecretHash parameter, see [How do I troubleshoot "Unable to verify secret hash for client <client-id>" errors from my Amazon Cognito user pools API?](#) in the AWS Knowledge Center.

You can use the following code examples in your server-side app code.

Shell

```
echo -n "[username][app client ID]" | openssl dgst -sha256 -hmac [app client secret]
-binary | openssl enc -base64
```

Java

```
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 ");
    }
}
```

Python

```
import sys
import hmac, hashlib, base64
username = sys.argv[1]
app_client_id = sys.argv[2]
key = sys.argv[3]
message = bytes(sys.argv[1]+sys.argv[2], 'utf-8')
key = bytes(sys.argv[3], 'utf-8')
secret_hash = base64.b64encode(hmac.new(key, message,
    digestmod=hashlib.sha256).digest()).decode()
print("SECRET HASH:", secret_hash)
```

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, 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 updates their email address or phone number in your app, Amazon Cognito immediately sends a message with a verification code to a user if you configured your user pool

to automatically verify that attribute. The user must then provide the code from the verification message to your app. Your app then submits the code in a [VerifyUserAttribute](#) API request to complete verification of the new attribute value.

If your user pool doesn't require that users verify an updated email address or phone number, Amazon Cognito immediately changes the value of an updated `email` or `phone_number` attribute and marks the attribute as unverified. Your user can't sign in with an unverified email or phone number. They must complete verification of the updated value before they can use that attribute as a sign-in alias.

If your user pool requires that users verify an updated email address or phone number, Amazon Cognito keeps the attribute verified and set to its original value until your user verifies the new attribute value. If the attribute is an alias for sign-in, your user can sign in with the original attribute value until verification changes the attribute to the new value. For more information about how to configure your user pool to require users to verify updated attributes, see [Configuring email or phone verification](#).

You can use a custom message Lambda trigger to customize the verification message. For more information, see [Custom message Lambda trigger](#). When a user's email address or phone number is unverified, your app should inform the user that they must verify the attribute, and provide a button or link for users to verify their new email address 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 [Customize email and SMS messages](#) in [Creating user accounts as administrator](#) and the Custom Message trigger in [Customizing user pool workflows with Lambda triggers](#).

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](#)) 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](#).

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 sends email messages to 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.

Configuring email or phone verification

You can choose settings for email or phone verification under the **Messaging** menu. For more information on multi-factor authentication (MFA), see [SMS Text Message MFA](#).

Amazon Cognito uses Amazon SNS to send SMS messages. If you haven't sent an SMS message from Amazon Cognito or any other AWS service before, Amazon SNS might place your account in the SMS sandbox. We recommend that you send a test message to a verified phone number before you remove your account from the sandbox to production. Additionally, if you plan to send SMS messages to US destination phone numbers, you must obtain an origination or Sender ID from Amazon Pinpoint. To configure your Amazon Cognito user pool for SMS messages, see [SMS message settings for Amazon Cognito user pools](#).

Amazon Cognito can automatically verify email addresses or phone numbers. To do this verification, Amazon Cognito sends a verification code or a verification link. For email addresses, Amazon Cognito can send a code or a link in an email message. You can choose a **Verification type** of **Code** or **Link** when you edit your **Verification message** template in the **Messaging** menu under **Branding** in the Amazon Cognito console. For more information, see [Customizing email verification messages](#).

For phone numbers, Amazon Cognito sends a code in an SMS text message.

Amazon Cognito must verify a phone number or email address to confirm users and help them to recover forgotten passwords. Alternatively, you can automatically confirm users with the pre sign-up Lambda trigger or use the [AdminConfirmSignUp](#) API operation. For more information, see [Signing up and confirming user accounts](#).

The verification code or link is valid for 24 hours.

If you choose to require verification for an email address or phone number, Amazon Cognito automatically sends the verification code or link when a user signs up. If the user pool has a [Custom SMS sender Lambda trigger](#) or [Custom email sender Lambda trigger](#) configured, that function is invoked instead.

Notes

- Amazon SNS charges separately for SMS text messaging that it uses to verify phone numbers. There is no charge to send email messages. 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 generate email messages from Amazon Cognito, use a real email address that Amazon Cognito can reach without hard bounces. For more information, see [the section called “Sending emails while testing your app”](#).
- The forgotten password flow requires either the user's email or the user's phone number to verify the user.

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, Amazon Cognito sends a verification code to the phone number through SMS message. Amazon Cognito hasn't yet verified the email address, so your app must call [GetUser](#) to see if an email address awaits verification. If it does require verification, the app must call [GetUserAttributeVerificationCode](#) to initiate the email verification flow. Then it must submit the verification code by calling [VerifyUserAttribute](#).

You can adjust your SMS message spend quota for an AWS account and for individual messages. The limits apply only to the cost to send SMS messages. For more information, see **What are account-level and message-level spend quotas and how do they work?** in the [Amazon SNS FAQs](#).

Amazon Cognito sends SMS messages using Amazon SNS resources in either the AWS Region where you created the user pool or in a **Legacy Amazon SNS alternate Region** from the following table. The exception is Amazon Cognito user pools in the Asia Pacific (Seoul) Region. These user pools use your Amazon SNS configuration in the Asia Pacific (Tokyo) Region. For more information, see [Choose the AWS Region for Amazon SNS SMS messages](#).

Amazon Cognito Region	Legacy Amazon SNS alternate Region
US East (Ohio)	US East (N. Virginia)
Asia Pacific (Mumbai)	Asia Pacific (Singapore)
Asia Pacific (Seoul)	Asia Pacific (Tokyo)
Canada (Central)	US East (N. Virginia)
Europe (Frankfurt)	Europe (Ireland)
Europe (London)	Europe (Ireland)

Example: If your Amazon Cognito user pool is in Asia Pacific (Mumbai), and you have increased your spend limit in ap-southeast-1, you might not want to request a separate increase in ap-south-1. Instead, you can use your Amazon SNS resources in Asia Pacific (Singapore).

Verifying updates to email addresses and phone numbers

An email address or phone number attribute can become active and unverified immediately after your user changes its value. Amazon Cognito can also require that your user verifies the new value before Amazon Cognito updates the attribute. When you require that your users first verify the new value, they can use the original value for sign-in and to receive messages until they verify the new value.

When your users can use their email address or phone number as a sign-in alias in your user pool, their sign-in name for an updated attribute depends on whether you require verification of updated attributes. When you require that users verify an updated attribute, a user can sign in with

the original attribute value until they verify the new value. When you don't require that users verify an updated attribute, a user can't sign in or receive messages at either the new or the original attribute value until they verify the new value.

For example, your user pool allows sign-in with an email address alias, and requires that users verify their email address when they update. Sue, who signs in as sue@example.com, wants to change her email address to sue2@example.com but accidentally enters ssue2@example.com. Sue doesn't receive the verification email, so she can't verify ssue2@example.com. Sue signs in as sue@example.com and resubmits the form in your app to update her email address to sue2@example.com. She receives this email, provides the verification code to your app, and begins signing in as sue2@example.com.

When a user updates an attribute and your user pool verifies new attribute values

- They can sign in with the original attribute value before they have confirmed the code to verify the new value.
- They can only sign in with the new attribute value after they have confirmed the code to verify the new value.
- If you set `email_verified` or `phone_number_verified` to `true` in an [AdminUpdateUserAttributes](#) API request, they can sign in before they have confirmed the code that Amazon Cognito sent to them.

When a user updates an attribute and your user pool doesn't verify new attribute values

- They can't sign in with, or receive messages at, the original attribute value.
- They can't sign in with, or receive messages other than a confirmation code at, the new attribute value before they have confirmed the code to verify the new value.
- If you set `email_verified` or `phone_number_verified` to `true` in an [AdminUpdateUserAttributes](#) API request, they can sign in before they have confirmed the code that Amazon Cognito sent to them.

To require attribute verification when users update their email address or phone number

1. Sign in to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.

3. In the **Sign-up** menu, choose **Edit** under **Attribute verification and user account confirmation**.
4. Choose **Keep original attribute value active when an update is pending**.
5. Under **Active attribute values when an update is pending**, choose the attributes that you want to require your users verify before Amazon Cognito updates the value.
6. Choose **Save changes**.

To require attribute update verification with the Amazon Cognito API, you can set the `AttributesRequireVerificationBeforeUpdate` parameter in an [UpdateUserPool](#) request.

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 **Messaging** menu of the Amazon Cognito console under **SMS**, choose **Edit** to set a role.

Configuring verification and invitation messages

With Amazon Cognito, you can customize SMS and email verification messages and user invitation messages, to enhance the security and user experience of your application. With Amazon Cognito, you can choose between code-based or one-click link verifications to suit your application's needs. This topic discusses how you can personalize multi-factor authentication (MFA) and verification communications in the Amazon Cognito console.

In the **Messaging** menu under **Message templates**, you can customize:

- Your SMS text message multi-factor authentication (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. Similarly, the SMS MFA message template only appears if the MFA setting is **required** or **optional**.

Topics

- [Message templates](#)
- [Customizing the SMS message](#)
- [Customizing email verification messages](#)
- [Customizing user invitation messages](#)
- [Customizing your email address](#)
- [Authorizing Amazon Cognito to send Amazon SES email on your behalf \(from a custom FROM email address\)](#)

Message templates

You can use message templates to insert placeholders into your messages. Amazon Cognito replace the placeholders with the corresponding values. You can reference *Universal template placeholders* in message templates of any type, although these values won't be present in all message types.

Universal template placeholders

Description	Token	Message type
Verification code	{#####}	Verification, confirmation, and MFA messages
Temporary password	{#####}	Forgot-password and invitation messages
User name	{username}	Invitation and advanced security messages

One of the available automated responses with [threat protection](#) is to notify the user that Amazon Cognito detected potentially-malicious activity. You can use advanced security template placeholders to do the following:

- Include specific details about an event such as IP address, city, country, sign-in time, and device name. Amazon Cognito advanced security features can analyze these details.
- Verify whether a one-click link is valid.
- Use event ID, feedback token, and user name to build your own one-click link.

 **Note**

To generate one-click links and use the {one-click-link-valid} and {one-click-link-invalid} placeholders in advanced security email templates, you must already have a domain configured for your user pool.

Advanced security features add the following placeholders that you can insert into message templates:

Advanced security template placeholders

Description	Token
IP address	{ip-address}
City	{city}
Country	{country}
Log-in time	{login-time}
Device name	{device-name}
One-click link is valid	{one-click-link-valid}
One-click link is not valid	{one-click-link-invalid}
Event ID	{event-id}

Description	Token
Feedback token	{feedback-token}

Customizing the SMS message

 **Note**

In the new Amazon Cognito console experience, you can customize SMS messages

You can customize the SMS message for multi-factor authentication (MFA) in the **Messaging** menu under the **Message templates** heading.

 **Important**

Your custom message must contain the {#####} placeholder. This placeholder is replaced with the authentication code before the message is sent.

Amazon Cognito imposes a maximum length for SMS messages, including the authentication code, at 140 UTF-8 characters.

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. This placeholder is replaced with the verification code before the message is sent.

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

Customizing email verification messages

To verify the email address of a user in your user pool with Amazon Cognito, you can send the user an email message with a link that they can select, or you can send them a code that they can enter.

To customize the email subject and message content for email address verification messages, edit the **Verification message** template in the **Messaging** menu of your user pool. You can choose a **Verification type** of **Code** or **Link** when you edit your **Verification message** template.

When you choose **Code** as the verification type, your custom message must contain the `{#####}` placeholder. When you send the message, the verification code replaces this placeholder.

When you choose **Link** as the verification type, your custom message must include a placeholder in the format `{##Verify Your Email##}`. You can change the text string between the placeholder characters, for example `{##Click here##}`. A verification link titled *Verify Your Email* replaces this placeholder.

The link for an email verification message directs your user to a URL like the following example.

```
https://<your user pool domain>/confirmUser/?  
client_id=abcdefg12345678&user_name=emailtest&confirmation_code=123456
```

The maximum length for the message, including the verification code (if present), is 20,000 UTF-8 characters. You can use HTML tags in this message to format the contents.

Customizing user invitation messages

You can customize the user invitation message that Amazon Cognito sends to new users by SMS or email message by editing the **Invitation messages** template in the **Messaging** menu.

Important

Your custom message must contain the `{username}` and `{#####}` placeholders. When Amazon Cognito sends the invitation message, it replaces these placeholders with your user's user name and password.

The maximum length of an SMS message, including the verification code, is 140 UTF-8 characters. The maximum length of an email message, including the verification code, is 20,000 UTF-8 characters. You may use HTML tags in your email messages to format the contents.

Customizing your email address

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

To customize the FROM and REPLY-TO email addresses

1. Navigate to the [Amazon Cognito console](#), and choose **User Pools**.
2. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Messaging** menu. Under **Email**, choose **Edit**.
4. Choose an **SES Region**.
5. Choose a **FROM email address** from the list of email addresses you have verified with Amazon SES in the **SES Region** you selected. To use an email address from a verified domain, configure email settings in the AWS Command Line Interface or the AWS API. For more information, see [Verifying email addresses and domains in Amazon SES](#) in the *Amazon Simple Email Service Developer Guide*.
6. Choose a **Configuration set** from the list of configuration sets in your chosen **SES Region**.
7. Enter a friendly **FROM sender name** for your email messages, in the format John Stiles <johnstiles@example.com>.
8. 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)

You can configure Amazon Cognito to send email from a custom FROM email address instead of its default address. To use a custom address, you must give Amazon Cognito permission to send email message from an Amazon SES verified identity. In most cases, you can grant permission by creating a sending authorization policy. For more information, see [Using sending authorization with Amazon SES](#) in the *Amazon Simple Email Service Developer Guide*.

When you configure a user pool to use Amazon SES for email messages, Amazon Cognito creates the `AWSServiceRoleForAmazonCognitoIdpEmailService` role in your account to grant access to Amazon SES. No sending authorization policy is needed when the `AWSServiceRoleForAmazonCognitoIdpEmailService` service-linked role is used. You only

need to add a sending authorization policy when you use both the default email functionality in your user pool *and* a verified Amazon SES identity as the FROM address.

For more information about the service-linked role that Amazon Cognito creates, see [Using service-linked roles for Amazon Cognito](#).

The following example sending authorization policy grants Amazon Cognito a limited ability to use an Amazon SES verified identity. Amazon Cognito can only send email messages when it does so on behalf of both the user pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition. 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": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "stmnt1234567891234",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "email.cognito-idp.amazonaws.com"  
                ]  
            },  
            "Action": [  
                "SES:SendEmail",  
                "SES:SendRawEmail"  
            ],  
            "Resource": "<your SES identity ARN>",  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceAccount": "<your account number>"  
                },  
                "ArnLike": {  
                    "aws:SourceArn": "<your user pool ARN>"  
                }  
            }  
        }  
    ]  
}
```

```
        }
    }
}
]
```

The Amazon Cognito console adds a similar 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 a policy structured like the previous example to your Amazon SES Identity.

Creating user accounts as administrator

User pools aren't only a customer identity and access management (CIAM) user directory, where anyone on the internet can sign up for a user profile in your application. You can disable self-service sign-up. You might already know your customers and want to only admit those who have been authorized in advance. You can put manual authentication guardrails around your application with a [private SAML 2.0 or OIDC identity provider](#), by [importing users](#), by [screening users at sign-up](#)—or by creating users with administrative API operations. Your workflow for administrative creation of users can be programmatic, provisioning users after they register in another system, or it can be on a case-by-case or testing basis in the Amazon Cognito console.

When you create users as an administrator, Amazon Cognito sets a temporary password for them and sends a welcome, or invitation, message. They can follow the link in their invitation message and sign in for the first time, setting a password and confirming their account. The page that follows describes how to create new users and configure the welcome message. For more information about user creation with the user pools API and an AWS SDK or CDK, see [AdminCreateUser](#).

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.

The following are some examples of how administrators can manage users in user pools.

- Create a new user profile in the Amazon Cognito console or with the `AdminCreateUser` API operation.
- Make username-and-password, passwordless, passkey, and custom [authentication flows](#) available to your user pool and app client.

- Set user attribute values.
- Create custom attributes.
- Set the value of immutable [custom attributes](#) in AdminCreateUser API requests. This feature isn't available in the Amazon Cognito console.
- Specify a temporary password, create a user without a password, or allow Amazon Cognito to automatically generate a password.
- Create new users and automatically confirm their accounts, verify their email addresses, or verify their phone numbers.
- Specify custom SMS and email invitation messages for new users via the AWS Management Console or Lambda triggers like [custom message](#), [custom SMS sender](#), and [custom email sender](#).
- 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.
- [Suppress](#) the sending of the invitation message when the user is created.
- Specify an expiration time limit of up to 90 days for new user accounts.
- Allow users to sign themselves up or require that new users only be added by the administrator.

Administrators can also sign users in with AWS credentials in a server-side application. For more information, see [Authorization models for API and SDK authentication](#).

User authentication flows and creating users

Administrative creation of users has options that differ based on the configuration of your user pool. The *authentication flows*, or methods available to users for sign-in and MFA, can change how you create users and the messages that you send to them. The following are some authentication flows that are available in user pools.

- Username and password
- Passkeys
- Sign-in with third-party IdPs
- Passwordless with email and SMS one-time passwords (OTPs)
- Multi-factor authentication with email, SMS, and authenticator-app OTPs
- Custom authentication with Lambda triggers

For more information about how to configure these sign-in factors, see [Authentication with Amazon Cognito user pools](#). Here's what you need to know about the effect of this configuration on creation of user accounts.

You can create users without passwords

If you have enabled passwordless sign-in for your user pool, you can create users without passwords. To create a user without a password, you must provide attribute values for an available passwordless sign-in factor. For example, if email OTP passwordless sign-in is available in your user pool, you can create a user with no password and an email address attribute. If the only authentication flows available to new users require a password, for example passkey or username-password, you must create or generate a temporary password for each new user.

To create a new user without a password

- Choose **Don't set a password** in the Amazon Cognito console
- Omit or leave blank the `TemporaryPassword` parameter of your `AdminCreateUser` API request

Users without passwords are automatically confirmed

Normally new users get a temporary password and go into a `FORCE_CHANGE_PASSWORD` status when you create them. When you create users without passwords, they immediately go into a `CONFIRMED` state. You can't resend confirmation codes to these users in the `CONFIRMED` state.

Invitation messages change for users without passwords.

By default, Amazon Cognito sends an [invitation message](#) to new users that says Your `username` is `{userName}` and your `password` is `{#####}`. When you create users with no password, the message says Your `username` is `{userName}`. Customize your invitation message to reflect whether you will set passwords for users. Omit out the `{#####}` password variable in passwordless authentication models.

You can't autogenerate passwords when passwordless factors are available

If you have configured your user pool to support email or phone OTP passwordless sign-in, you must specify temporary passwords for new users.

Creating a new user in the AWS Management Console

You can set user password requirements, configure the invitation and verification messages sent to users, and add new users with the Amazon Cognito console.

Set a password policy and enable self-registration

You can configure settings for minimum password complexity and whether users can sign up using public APIs in your user pool.

Configure a password policy

1. Navigate to the [Amazon Cognito console](#), and choose **User Pools**.
2. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Authentication methods** menu and locate **Password policy**. Choose **Edit**.
4. Choose a **Password policy mode of Custom**.
5. Choose a **Password minimum length**. For limits to the password length requirement, see [User pools resource quotas](#).
6. Choose a **Password complexity** requirement.
7. Choose how long password set by administrators should be valid for.
8. Choose **Save changes**.

Allow self-service sign-up

1. Navigate to the [Amazon Cognito console](#), and choose **User Pools**.
2. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Sign-up** menu and locate **Self-service sign-up**. Select **Edit**.
4. Choose whether to **Enable self-registration**. Self-registration is typically used with public app clients that need to register new users in your user pool without distributing a client secret or AWS Identity and Access Management (IAM) API credentials.

Disabling self-registration

If you do not enable self-registration, new users must be created by administrative API actions using IAM API credentials or by sign-in with federated providers.

5. Choose **Save changes**.

Customize email and SMS messages

Customize user messages

You can customize the messages that Amazon Cognito sends to your users when you invite them to sign in, they sign up for a user account, or they sign in and are prompted for multi-factor authentication (MFA).

Note

An **Invitation message** is sent when you create a user in your user pool and invite them to sign in. Amazon Cognito sends initial sign-in information to the user's email address or phone number.

A **Verification message** is sent when a user signs up for a user account in your user pool. Amazon Cognito sends a code to the user. When the user provides the code to Amazon Cognito, they verify their contact information and confirm their account for sign-in. Verification codes are valid for 24 hours.

An **MFA message** is sent when you enable SMS MFA in your user pool, and a user that has configured SMS MFA signs in and is prompted for MFA.

1. Navigate to the [Amazon Cognito console](#), and choose **User Pools**.
2. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Messaging** menu and locate **Message templates**. Select **Verification messages**, **Invitation messages**, or **MFA messages** and choose **Edit**.
4. Customize the messages for the chosen message type.

Note

All variables in message templates must be included when you customize the message. If the variable, for example `{#####}`, is not included, your user will have insufficient information to complete the message action.

For more information, see [Message templates](#).

5. a. **Verification messages**

- i. Choose a **Verification type** for **Email** messages. A **Code** verification sends a numeric code that the user must enter. A **Link** verification sends a link the user can click to verify their contact information. The text in the variable for a **Link** message is displayed as hyperlink text. For example, a message template using the variable {##Click here##} is displayed as [Click here](#) in the email message.
 - ii. Enter an **Email subject** for **Email** messages.
 - iii. Enter a custom **Email message** template for **Email** messages. You can customize this template with HTML.
 - iv. Enter a custom **SMS message** template for **SMS** messages.
 - v. Choose **Save changes**.
- b. **Invitation messages**
 - i. Enter an **Email subject** for **Email** messages.
 - ii. Enter a custom **Email message** template for **Email** messages. You can customize this template with HTML.
 - iii. Enter a custom **SMS message** template for **SMS** messages.
 - iv. Choose **Save changes**.
 - c. **MFA messages**
 - i. Enter a custom **SMS message** template for **SMS** messages.
 - ii. Choose **Save changes**.

Create a user

Create a user

You can create new users for your user pool from the Amazon Cognito console. Typically, users can sign in after they set a password. To sign in with an email address, a user must verify the email attribute. To sign in with a phone number, the user must verify the phone_number attribute. To confirm accounts as an administrator, you can also use the AWS CLI or API, or create user profiles with a federated identity provider. For more information, see the [Amazon Cognito API Reference](#).

1. Navigate to the [Amazon Cognito console](#), and choose **User Pools**.
2. Choose an existing user pool from the list, or [create a user pool](#).
3. Choose the **Users** menu, and choose **Create a user**.

4. Review the **User pool sign-in and security requirements** for guidance on password requirements, available account recovery methods, and alias attributes for your user pool.
5. Choose how you want to send an **Invitation message**. Choose SMS message, email message, or both. To suppress the invitation message, choose **Don't send an invitation**.

 **Note**

Before you can send invitation messages, configure a sender and an AWS Region with Amazon Simple Notification Service and Amazon Simple Email Service in the **Messaging** menu of your user pool . Recipient message and data rates apply. Amazon SES bills you for email messages separately, and Amazon SNS bills you for SMS messages separately.

6. Choose a **Username** for the new user.
7. Choose if you want to **Create a password** or have Amazon Cognito **Generate a password** for the user. The option to generate a password isn't available if [passwordless sign-in](#) is available in the user pool. Any temporary password must adhere to the user pool password policy.
8. Choose **Create**.
9. Choose the **Users** menu and choose the **User name** entry for the user. Add and edit **User attributes** and **Group memberships**. Review **User event history**.

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.

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's 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 access and ID tokens, 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 belongs 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, the group with the lowest precedence value will have its IAM role 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 both the ID token and access token from a user pool in the `cognito:groups` claim. You can submit ID or access tokens with requests to Amazon API Gateway and use an Amazon Cognito user pool authorizer for a REST API. For more information, see [Control access to a REST API using Amazon Cognito user pools as authorizer](#) in the [API Gateway Developer Guide](#).

You can also authorize access to an Amazon API Gateway HTTP API with a custom JWT authorizer. For more information, see [Controlling access to HTTP APIs with JWT authorizers](#) in the [API Gateway Developer Guide](#).

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 quotas](#).
- Groups cannot be nested.
- You cannot search for users in a group.
- You cannot search for groups by name, but you can list groups.

Creating a new group in the AWS Management Console

Use the following procedure to create a new group.

To create a new group

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.

2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Groups** menu, and then choose **Create a group**.
5. On the **Create a group** page, in **Group name**, enter a friendly name for your new group.
6. You can optionally provide additional information about this group using any of the following fields:
 - **Description** - Enter details about what this new group will be used for.
 - **Precedence** - Amazon Cognito evaluates and applies all group permissions for a given user based on which groups that they belong to has a lower precedence value. The group with the lower precedence will be chosen and its associated IAM role will be applied. For more information, see [Assigning precedence values to groups](#).
 - **IAM role** - You can assign an IAM role to your group when you need to control permissions to your resources. 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. For more information, see [Assigning IAM roles to groups](#).
 - **Add users to this group** - Add existing users as members of this group after it is created.
7. Choose **Create** to confirm.

Managing and searching for user accounts

Users pools can contain millions of users. Working with a dataset of this size is a challenge for administrators. Amazon Cognito has tools for finding and modifying user profiles. The top methods for finding users are the **Users** menu of the Amazon Cognito console, and with [ListUsers](#). Of the methods that retrieve information about users, these are the options that don't have a cost impact unlike, for example, [Admin GetUser](#).

This section of the guide has information about finding and updating user profiles in a user pool.

Viewing user attributes

Use the following procedure to view user attributes in the Amazon Cognito console.

To view user attributes

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.

2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select a user in the list.
5. On the user details page, under **User attributes**, you can view which attributes are associated with the user.

Resetting a user's password

Use the following procedure to reset a user's password in the Amazon Cognito console.

To reset a user's password

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select a user in the list.
5. On the user details page, choose **Actions, Reset password**.
6. In the **Reset password** dialog, review the information and when ready, choose **Reset**.

This action immediately results in a confirmation code being sent to the user and disables the user's current password by changing the user state to RESET_REQUIRED. The **Reset password** code is valid for 1 hour.

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

Note

You can also list users with a client-side filter. The server-side filter matches no more than 1 attribute. For advanced search, use a client-side filter with the --query parameter of the `list-users` action in the AWS Command Line Interface. When you use a client-side filter, `ListUsers` returns a paginated list of zero or more users. You can receive multiple pages in a row with zero results. Repeat the query with each pagination token that is returned until you receive a null pagination token value, then review the combined result.

For more information about server-side and client-side filtering, see [Filtering AWS CLI output](#) in the AWS Command Line Interface User Guide.

Searching for users with 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.

To search for a user in the Amazon Cognito console

1. Go to the [Amazon Cognito console](#). You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and enter the username in the search field. Note that some attribute values are case-sensitive (for example, **Username**).

You can also find users by adjusting the search filter to narrow the scope down to other user properties, such as **Email**, **Phone number**, or **Last name**.

Searching for users with 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. To retrieve all attributes, don't include an `AttributesToGet` parameter or request `AttributesToGet` with a value of the literal string `null`.
- `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. Amazon Cognito expires the pagination token after one hour.
- `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": null,  
    "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": null,  
    "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"
}
```

Passwords, account recovery, and password policies

All users who sign in to a user pool, even [federated users](#), have passwords assigned to their user profiles. [Local users](#) and [linked users](#) must provide a password when they sign in. Federated users don't use user pool passwords, but sign in with their identity provider (IdP). You can permit users to reset their own passwords, reset or change passwords as an administrator, and [set policies](#) for password complexity and history.

Amazon Cognito doesn't store user passwords in plaintext. Instead, it stores a hash of each user's password with a user-specific salt. Because of this, you can't retrieve existing passwords from the user profiles in your user pools. As a best practice, don't store plaintext user passwords anywhere. Perform password resets when users forget their passwords.

Password reset and recovery

Users forget their passwords. You might want them to be able to reset their password themselves, or you might want to require that an administrator resets their password for them. Amazon Cognito user pools have options for both models. This part of the guide covers the user pool settings and the API operations for password reset.

The [ForgotPassword](#) API operation and the managed login option **Forgot your password?** send users a code that, when they confirm that they have the correct code, gives them an opportunity to set a new password with [ConfirmForgotPassword](#). This is the self-service password-recovery model.

The [AdminSetUserPassword](#) and [AdminResetUserPassword](#) API operations are the administrator-initiated methods of password reset. AdminSetUserPassword sets a temporary or permanent password, and AdminResetUserPassword sends users a password-reset code in the same way as ForgotPassword.

The AccountRecoverySetting parameter is the user pool parameter that sets the methods that users 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. The recovery code is valid for one hour. When you specify an [AccountRecoverySetting](#) for your user pool, Amazon Cognito chooses the code delivery destination based on the priority that you set.

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.

A user's preferred MFA method influences the methods they can use to recover their password. Users whose preferred MFA is by email message can't receive a password-reset code by email. Users whose preferred MFA is by SMS message can't receive a password-reset code by SMS.

Your [password recovery](#) settings must provide an alternative option when users aren't eligible for your preferred password-reset method. For example, your recovery mechanisms might have email as first priority and email MFA might be an option in your user pool. In this case, add SMS-message account recovery as a second option or use administrative API operations to reset passwords for those users.

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.

Adding user pool password requirements

Strong, complex passwords are a security best practice for your user pool. Especially in applications that are open to the internet, weak passwords can expose your users' credentials to systems that guess passwords and try to access your data. The more complex a password is, the more difficult

it is to guess. Amazon Cognito has additional tools for security-conscious administrators, like [advanced security features](#) and [AWS WAF web ACLs](#), but your password policy is a central element of the security of your user directory.

Passwords for local users in Amazon Cognito user pools don't automatically expire. As a best practice, log the time, date, and metadata of user password resets in an external system. With an external log of password age, your application or a Lambda trigger can look up a user's password age and require a reset after a given period.

You can configure your user pool to require a minimum password complexity that conforms to your security standards. Complex passwords have a minimum length of at least eight characters. They also include a mix of uppercase, numeric, and special characters.

With advanced security features, you can also set a policy for password reuse. You can prevent a user from resetting their password to a new password that matches their current password or any of up to 23 additional previous passwords, for a maximum total of 24.

To set a user pool password policy

1. Create a user pool and navigate to the **Configure security requirements** step, or access an existing user pool and navigate to the **Authentication methods** menu.
2. Navigate to **Password policy**.
3. Choose a **Password policy mode**. **Cognito defaults** configures your user pool with the recommended minimum settings. You can also choose a **Custom** password policy.
4. Set a **Password minimum length**. All users must sign up or be created with a password whose length is greater than or equal to this value. You can set this minimum value as high as 99, but your users can set passwords up to 256 characters long.
5. Configure password complexity rules under **Password requirements**. Choose the character types—numbers, special characters, uppercase letters, and lowercase letters—that you want to require at least one of in each user's password.

You can require at least one of the following characters in passwords. After Amazon Cognito verifies that passwords contain the minimum required characters, your users' passwords can contain additional characters of any type up to the maximum password length.

- Uppercase and lowercase [basic latin](#) letters
- Numbers
- The following special characters.

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

- Non-leading, non-trailing space characters.
6. Set a value for **Temporary passwords set by administrators expire in**. After this period has passed, a new user that you created in the Amazon Cognito console or with `AdminCreateUser` can't sign in and set a new password. After they sign in with their temporary password, their user accounts never expire. To update the password duration in the Amazon Cognito user pools API, set a value for [TemporaryPasswordValidityDays](#) in your [CreateUserPool](#) or [UpdateUserPool](#) API request.
 7. Set a value for **Prevent use of previous passwords**, if available. To use this feature, activate [advanced security features](#) in your user pool. The value of this parameter is the number of previous passwords that a new password is prevented from matching when a user resets their password.

To reset access for an expired user account, do one of the following:

- Delete the user profile and create a new one.
- Set a new permanent password in an [AdminSetUserPassword](#) API request.
- Generate a new confirmation code in an [AdminResetUserPassword](#) API request.

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.

More resources

- [Approaches for migrating users to Amazon Cognito user pools](#)
- [AWS re:Inforce 2023 - Migrating to Amazon Cognito](#)

Topics

- [Importing users with a user migration Lambda trigger](#)
- [Importing users into user pools from a CSV file](#)

Importing users with a user migration Lambda trigger

With this approach, you can seamlessly migrate users from your existing user directory to user pools when a user signs in for the first time with your app or requests a password reset. Add a [Migrate user Lambda trigger](#) function to your user pool and it receives metadata about users who try to sign in, and returns user profile information from an external identity source. For details and example code for this Lambda trigger, including request and response parameters, see [Migrate user Lambda trigger parameters](#).

Before you start to migrate users, create a user migration Lambda function in your AWS account, and set the Lambda function as the user migration trigger in your user pool. Add an authorization policy to your Lambda function that permits only the Amazon Cognito service account principal, `cognito-idp.amazonaws.com` to invoke the Lambda function, and only in the context of your own user pool. For more information, see [Using resource-based policies for AWS Lambda \(Lambda function policies\)](#).

Sign-in process

1. The user opens your app and signs in with the Amazon Cognito user pools API or through managed login. For more information about how to facilitate sign-in with Amazon Cognito APIs, see [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#).
2. Your app sends the user name and password to Amazon Cognito. If your app has a custom sign-in UI that you built with an AWS SDK, your app must use [InitiateAuth](#) or [AdminInitiateAuth](#) with the `USER_PASSWORD_AUTH` or `ADMIN_USER_PASSWORD_AUTH` flow. When your app uses one of these flows, the SDK sends the password to the server.

Note

Before you add a user migration trigger, activate the `USER_PASSWORD_AUTH` or `ADMIN_USER_PASSWORD_AUTH` flow in the settings of your app client. You must use these flows instead of the default `USER_SRP_AUTH` flow. Amazon Cognito must send a password to your Lambda function so that it can verify your user's authentication in the other directory. An SRP obscures your user's password from your Lambda function.

3. Amazon Cognito checks if the submitted user name matches a user name or alias in the user pool. You can set the user's email address, phone number, or preferred user name as an alias in your user pool. If the user doesn't exist, Amazon Cognito sends parameters, including the user name and password, to your [Migrate user Lambda trigger](#) function.
4. Your [Migrate user Lambda trigger](#) function checks for or authenticates the user with your existing user directory or user database. The function returns user attributes that Amazon Cognito stores in the user's profile in the user pool. You can return a `username` parameter only if the submitted user name matches an alias attribute. If you want users to continue to use their existing passwords, your function sets the attribute `finalUserStatus` to CONFIRMED in the Lambda response. Your app must return all "response" parameters shown at [Migrate user Lambda trigger parameters](#).

 **Important**

Do not log the entire request event object in your user migration Lambda code. This request event object includes the user's password. If you don't sanitize the logs, passwords appear in CloudWatch Logs.

5. Amazon Cognito creates the user profile in your user pool, and returns tokens to your app client.
6. Your app performs token intake, accepts the user authentication, and proceeds to the requested content.

After you migrate your users, use `USER_SRP_AUTH` for sign-in. The Secure Remote Password (SRP) protocol doesn't send the password across the network, and provides security benefits over the `USER_PASSWORD_AUTH` flow that you use during migration.

In case of errors during migration, including client device or network issues, your app receives error responses from the Amazon Cognito user pools API. When this happens, Amazon Cognito might or might not create the user account in your user pool. The user should then attempt to sign in again. If sign-in fails repeatedly, attempt to reset the user's password with the forgot-password flow in your app.

The forgot-password flow also invokes your [Migrate user Lambda trigger](#) function with a `UserMigration_ForgotPassword` event source. Because the user doesn't submit a password when they request a password reset, Amazon Cognito doesn't include a password in the event that it sends to your Lambda function. Your function can only look up the user in your existing user directory and return attributes to add to the user profile in your user pool. After your function

completes its invocation and returns its response to Amazon Cognito, your user pool sends a password reset code by email or SMS. In your app, prompt your user for their confirmation code and a new password, then send that information to Amazon Cognito in a [ConfirmForgotPassword](#) API request. You can also use the built-in pages for the forgot-password flow in managed login.

Additional resources

- [Approaches for migrating users to Amazon Cognito user pools](#)

Importing users into user pools from a CSV file

When you have an external identity store and the time to prepare your user pool for new local users, a bulk user import from a comma-separated values (CSV) file can be a low-effort, low-cost option for a migration to an Amazon Cognito user pool. A CSV file import is a process of downloading and populating a template file, then handing off the file to your user pool in an import job. You can use a CSV import to quickly create test users. You can also programmatically populate the file with read API requests to your external identity store, followed by parsing their details and attributes into write operations to the 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. Your users are in a RESET_REQUIRED state when imported using this method.

The lowest-effort way to import users from a CSV is to activate [passwordless sign-in](#) in your user pool. With email address and phone number attributes and the right user pool configuration, users can sign in with email or SMS one-time passwords (OTPs) immediately after your import job completes. For more information, see [Requiring imported users to reset their passwords](#).

You can also set your users' passwords with an [AdminSetUserPassword](#) API request that sets the Permanent parameter to true. CSV import doesn't contribute to the billed monthly active users (MAUs) in your user pool. However, password-reset operations do generate MAUs. To manage costs when you import large numbers of users with password who might not be immediately active, set up your application to prompt users for a new password when they sign in and receive the RESET_REQUIRED challenge.

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.

Steps to create a user import job

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.

More resources

- [Cognito User Profiles Export Reference Architecture](#) for exporting user accounts between user pools

Topics

- [Creating the CloudWatch Logs IAM role](#)
- [Creating the user import CSV file](#)
- [Creating and running the Amazon Cognito user pool import job](#)
- [Viewing the user pool import results in the CloudWatch console](#)
- [Requiring imported users to reset their passwords](#)

Creating the CloudWatch Logs IAM role

If you're using the Amazon Cognito CLI or API, then you need to create a CloudWatch IAM role. The following procedure describes how to create an IAM role that Amazon Cognito can use to write the results of your import job to CloudWatch Logs.

Note

When you create an import job in the Amazon Cognito console, you can create the IAM role at the same time. When you choose to **Create a new IAM role**, Amazon Cognito automatically applies the appropriate trust policy and IAM policy to the role.

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. Create a new IAM role for an AWS service. For detailed instructions, see [Creating a role for an AWS service](#) in the *AWS Identity and Access Management User Guide*.
 - a. When you select a **Use case** for your **Trusted entity type**, choose any service. Amazon Cognito isn't currently listed in service use cases.
 - b. In the **Add permissions** screen, choose **Create policy** and insert the following policy statement. Replace **REGION** with the AWS Region of your user pool, for example us-east-1. Replace **ACCOUNT** with your AWS account ID, for example 111122223333.

```
{  
    "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/*"  
            ]  
        }  
    ]  
}
```

3. Because you didn't choose Amazon Cognito as the trusted entity when you created the role, you now must manually edit the trust relationship of the role. Choose **Roles** from navigation pane of the IAM console, then choose the new role that you created.
4. Choose the **Trust relationships** tab.
5. Choose **Edit trust policy**.
6. Paste the following policy statement into **Edit trust policy**, replacing any existing text:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "cognito-idp.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

7. Choose **Update policy**.
8. Note the role ARN. You'll provide the ARN when you create your import job.

Creating the user import CSV file

Before you can import your existing users into your user pool, you must create a comma-separated values (CSV) file that contains the users that you want to import, and their attributes. From your user pool, you can retrieve a user import file with headers that reflect the attribute schema of your user pool. You can then insert user information that matches the formatting requirements in [Formatting the CSV file](#).

Downloading the CSV file header (console)

Use the following procedure to download the CSV header file.

To download the CSV file header

1. Go to the [Amazon Cognito console](#). You might be prompted for your AWS credentials.
2. Choose **User Pools**.

3. Choose an existing user pool from the list.
4. Choose the **Users** menu.
5. In the **Import users** section, choose **Create an import job**.
6. Under **Upload CSV**, select the *template.csv* link and download the 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-identity 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 the following string. It also includes any custom attributes you have added to your user pool.

```
cognito:username,name,given_name,family_name,middle_name,nickname,preferred_username,profile,pi
```

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 [Working with user attributes](#).

- The first row in the file is the downloaded header row, which contains the user attribute names.
- The order of columns in the CSV file doesn't matter.
- Each row after the first row 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**
 - At least one of the auto-verified attributes must be `true` for each user. An auto-verified attribute is an email address or phone number that Amazon Cognito automatically sends a code to when a new user joins your user pool.
 - The user pool must have at least one auto-verified attribute, either **email_verified** or **phone_number_verified**. 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

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 by SMS message. For more information, see [Verifying contact information at sign-up](#).

- **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
- 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 can't be empty.
- The maximum row length is 16,000 characters.
- The maximum CSV file size is 100 MB.
- The maximum number of rows (users) in the file is 500,000. This maximum doesn't include the header row.
- 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.

The following list is a example CSV import file for a user pool with no custom attributes. Your user pool schema might differ from this example. In that case, you must provide test values in the CSV template that you download from your user pool.

```
cognito:username, name, given_name, family_name, middle_name, nickname, preferred_username, profile, pi  
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 (AWS CLI).

Topics

- [Importing users from a CSV file \(console\)](#)
- [Importing users \(AWS CLI\)](#)

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. Go to the [Amazon Cognito console](#). You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu.
5. In the **Import users** section, choose **Create an import job**.
6. On the **Create import job** page, enter a **Job name**.
7. Choose to **Create a new IAM role** or to **Use an existing IAM role**.
 - a. If you chose **Create a new IAM role**, enter a name for your new role. Amazon Cognito will automatically create a role with the correct permissions and trust relationship. The IAM principal that creates the import job must have permissions to create IAM roles.
 - b. If you chose **Use an existing IAM role**, choose a role from the list under **IAM role selection**. This role must have the permissions and trust policy described in [Creating the CloudWatch Logs IAM role](#).
8. Choose **Create job** to submit your job, but start it later. Choose **Create and start job** to submit your job and start it immediately.

9. If you created your job but didn't start it, you can start it later. In the **Users** menu under **Import users**, choose your import job, then select **Start**. You can also submit a [StartUserImportJob API request](#) from an AWS SDK.
10. Monitor the progress of your user import job in the **Users** menu under **Import users**. If your job doesn't succeed, you can select the **Status** value. For additional details, select **View the CloudWatch logs for more details** and review any issues in the CloudWatch Logs console.

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 user pool ID for the user pool into which the new users will be added, and `ROLE_ARN` is the role ARN you received in [Creating the CloudWatch Logs IAM role](#):

```
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.

Example 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 can't 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"
```

Example 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
```

Example 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.

Example 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"
```

Example 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](#)
- [Interpreting the results](#)

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

If your user pool only offers password-based sign-in, users must reset their passwords after they are imported. The first time they sign in, they can enter *any* password. Amazon Cognito prompts them to enter a new password in the API response to the sign-in request from your application.

If your user pool has passwordless authentication factors, Amazon Cognito defaults to those for imported users. They're not prompted for a new password, and can sign in immediately with a passwordless email or SMS OTP. You can also prompt users to set a password so that they can complete other sign-in methods like username-password and passkey. The following conditions apply to passwordless sign-in after user import.

1. You must import users with an attribute that corresponds to an available passwordless sign-in factor. If users can sign in with an email address, you must import an email attribute. If a phone number, you must import a phone_number attribute. If both, import a value for either attribute.
2. Normally, users import in a RESET_REQUIRED state where they must reset their password. If they are imported with the ability to sign in with a passwordless factor, Amazon Cognito sets their state to CONFIRMED.

For more information about passwordless authentication including how to set it up and how to construct the authentication flow in your application, see [Authentication with Amazon Cognito user pools](#).

The following procedure describes the user experience in a custom-built login mechanism with local users in a RESET_REQUIRED after you import a CSV file. If your users sign in with managed login, Amazon Cognito prompts them to set a new password when they first sign in.

Requiring imported users to reset their passwords

1. In your app, silently attempt sign-in for the current user with `InitiateAuth` using a random password.
2. Amazon Cognito returns a `NotAuthorizedException` when `PreventUserExistenceErrors` is enabled. Otherwise, it returns `PasswordResetRequiredException`.
3. Your app makes a `ForgotPassword` API request and resets the user's password.
 - a. The app submits the username in a `ForgotPassword` API request.
 - b. Amazon Cognito sends a code to the verified email or phone number. The destination depends on the values you provided for `email_verified` and `phone_number_verified` in your CSV file. The response to the `ForgotPassword` request indicates the destination of the code.

 **Note**

Your user pool must be configured to verify emails or phone numbers. For more information, see [Signing up and confirming user accounts](#).

- c. Your app displays a message to your user to check the location where the code was sent, and prompts your user to enter the code and a new password.
- d. The user enters the code and new password in the app.
- e. The app submits the code and new password in a `ConfirmForgotPassword` API request.
- f. Your app redirects your user to sign-in.

Working with user attributes

Attributes are pieces of information that help you identify individual users, such as name, email address, and phone number. A new user pool has a set of default *standard attributes*. 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.

Don't store all information about your users in attributes. For example, keep user data that changes frequently, such as usage statistics or game scores, in a separate data store, such as Amazon Cognito Sync or Amazon DynamoDB.

 **Note**

Some documentation and standards refer to attributes as *members*.

Topics

- [Standard attributes](#)
- [Username and preferred username](#)
- [Customizing sign-in attributes](#)
- [Custom attributes](#)
- [Attribute permissions and scopes](#)

Standard attributes

Amazon Cognito assigns all users a set of standard attributes based on the [OpenID Connect specification](#). By default, standard and custom attribute values can be any string with a length of up to 2048 characters, but some attribute values have format restrictions.

The standard attributes are:

- name
- family_name
- given_name
- middle_name
- nickname
- preferred_username
- profile
- picture
- website
- gender
- birthdate

- zoneinfo
- locale
- updated_at
- address
- email
- phone_number
- sub

Except for sub, standard attributes are optional by default for all users. To make an attribute required, during the user pool creation process, select the **Required** check box next to the attribute. Amazon Cognito assigns a unique user identifier value to each user's sub attribute. Only the **email** and **phone_number** attributes can be verified.

Standard attributes have predefined properties that you can view in the SchemaAttributes parameter of a [DescribeUserPool API response](#). You can set custom values for these attribute properties, like data type, mutability, and length constraints. To modify standard attribute properties, set their custom values in the [CreateUserPool Schema parameter](#). The schema is also where you set required attributes. You can't modify the properties of standard attributes when you create user pools in the Amazon Cognito console.

 **Note**

When you mark a standard attribute as **Required**, a user can't register unless they provide a value for the attribute. To create users and not give values for required attributes, administrators can use the [AdminCreateUser](#) API. After you create a user pool, you can't switch an attribute between required and not required.

Standard attribute details and format restrictions

birthdate

Value must be a valid 10 character date in the format YYYY-MM-DD.

email

Users and administrators can verify email address values.

An administrator with proper AWS account permissions can change the user's email address and also mark it as verified. Mark an email address as verified with the [AdminUpdateUserAttributes API](#) or the [admin-update-user-attributes](#) AWS Command Line Interface (AWS CLI) command.

With this command, the administrator can change the `email_verified` attribute to `true`. You can also edit a user in the **Users** menu of the Amazon Cognito console to mark an email address as verified.

Value must be a valid email address string following the standard email format with @ symbol and domain, up to 2048 characters in length.

phone_number

A user must provide a phone number if SMS multi-factor authentication (MFA) is active. For more information, see [Adding MFA to a user pool](#).

Users and administrators can verify phone number values.

An administrator with proper AWS account permissions can change the user's phone number and also mark it as verified. Mark a phone number as verified with the [AdminUpdateUserAttributes API](#) or the [admin-update-user-attributes](#) AWS CLI command. With this command, the administrator can change the `phone_number_verified` attribute to `true`. You can also edit a user in the **Users** menu of the Amazon Cognito console to mark a phone number as verified.

⚠ Important

Phone numbers must follow these format 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. Remove any other characters from a phone number, such as parentheses, spaces, or dashes (-) before you submit the value to the service.

For example, a phone number based in the United States must follow this format:
+14325551212.

preferred_username

You can select `preferred_username` as required or as an alias, but not both. If the `preferred_username` is an alias, you can make a request to the [UpdateUserAttributes API](#) operation and add the attribute value after you confirm the user.

sub

Index and search your users based on the sub attribute. The sub attribute is a unique user identifier within each user pool. Users can change attributes like phone_number and email. The sub attribute has a fixed value. For more information about finding users, see [Managing and searching for user accounts](#).

View required attributes

Use the following procedure to view required attributes for a given user pool.

 **Note**

You can't change required attributes after you create a user pool.

To view required attributes

1. Go to [Amazon Cognito](#) in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Sign-up** menu.
5. In the **Required attributes** section, view the required attributes of your user pool.

Username and preferred username

The username value is a separate attribute and not the same as the name attribute. Each user has a username attribute. Amazon Cognito automatically generates a username for federated users. You must provide a username attribute to create a local user in the Amazon Cognito directory. After you create a user, you can't change the value of the username attribute.

Developers can use the preferred_username attribute to give users usernames that they can change. For more information, see [Customizing sign-in attributes](#).

If your application doesn't require a username, you don't need to ask users to provide one. Your app can create a unique username for users in the background. This can be useful if you want users

to register and sign in with an email address and password. For more information, see [Customizing sign-in attributes](#).

The `username` must be unique within a user pool. A `username` can be reused, but only after you delete it and it is no longer in use. For information about the string constraints to the `username` attributes, see the `username` property of a [SignUp](#) API request.

Customizing sign-in attributes

When you create a user pool, you can set up *username attributes* if you want your users to be able to sign up and sign in with an email address or phone number as their username. Alternatively, you can set up *alias attributes* to give your users the option: they can include multiple attributes when they sign up, and then sign in with a username, preferred username, email address, or phone number.

Important

After you create a user pool, you can't change this setting.

How to choose between alias attributes and username attributes

Your requirement	Alias attributes	Username attributes
Users have multiple sign-in attributes	Yes ¹	No ²
Users must verify email address or phone number before they can sign in with it	Yes	No
Sign up users with duplicate email addresses or phone numbers and prevent <code>UsernameExistsException</code> errors ³	Yes	No
Can assign the same email address or phone number	Yes ⁴	No

Your requirement	Alias attributes	Username attributes
attribute value to more than one user		

¹ Available sign-in attributes are username, email address, phone number, and preferred username.

² Can sign in with email address or phone number.

³ Your user pool doesn't generate `UsernameExistsException` errors when users register with potentially-duplicate email addresses or phone numbers, but no username. This behavior is independent of **Prevent username existence errors**, which applies to sign-in, but not sign-up, operations.

⁴ Only the last user who has verified the attribute can sign in with it.

Option 1: Multiple sign-in attributes (alias attributes)

You can activate aliases if you want to allow your users to choose to enter their username or other attribute values when they sign in. By default, users sign in with their username and password. The username is a fixed value that users can't change. If you mark an attribute as an alias, users can sign in with that attribute in place of the username. You can mark the email address, phone number, and preferred username attributes as aliases. For example, if you select email address and phone number as aliases for a user pool, users in that user pool can sign in with their username, email address, or phone number, along with their password.

To choose alias attributes, select **User name** and at least one additional sign-in option when you create your user pool.

Note

When you configure your user pool to be case insensitive, a user can use either lowercase or uppercase letters to sign up or sign in with their alias. For more information, see [CreateUserPool](#) in the *Amazon Cognito user pools API Reference*.

If you select email address as an alias, Amazon Cognito doesn't accept a username that matches a valid email address format. Similarly, if you select phone number as an alias, Amazon Cognito doesn't accept a username for that user pool that matches a valid phone number format.

Note

Alias values must be unique in a user pool. If you configure an alias for an email address or phone number, the value that you provide can be in a verified state in only one account. During sign-up, if your user provides an email address or phone number as an alias value and another user has already used that alias value, registration succeeds. However, when a user tries to confirm the account with this email (or phone number) and enters the valid code, Amazon Cognito returns an `AliasExistsException` error. The error indicates to the user that an account with this email address (or phone number) already exists. At this point, the user can abandon their attempt to create the new account and instead try to reset the password for the old account. If the user continues to create the new account, your app must call the `ConfirmSignUp` API with the `forceAliasCreation` option. `ConfirmSignUp` with `forceAliasCreation` moves the alias from the previous account to the newly created account, and marks the attribute unverified in the previous account.

Phone numbers and email addresses only become active aliases for a user after your user verifies the phone numbers and email addresses. We recommend that you choose automatic verification of email addresses and phone numbers if you use them as aliases.

Choose alias attributes to prevent `UsernameExistsException` errors for email address and phone number attributes when your users sign up.

Activate the `preferred_username` attribute so that your user can change the username that they use to sign in while their `username` attribute value doesn't change. If you want to set up 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 that they entered. If you select `preferred_username` as an alias, your user can provide the value only when they confirm an account. They can't provide the value during registration.

When the user signs up with a `username`, you can choose if they can sign in with one or more of the following aliases.

- Verified email address
- Verified phone number
- Preferred username

After the user signs up, they can change these aliases.

Important

When your user pool supports sign-in with aliases and you want to authorize or look up a user, don't identify your user by any of their sign-in attributes. The fixed-value user identifier sub is the only consistent indicator of your user's identity.

Include the following steps when you create the user pool so that users can sign in with an alias.

To configure a user pool so that users can sign in with a preferred username

1. Go to [Amazon Cognito](#) in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. In the top-right corner of the page, choose **Create a user pool** to start the user pool creation wizard.
4. In **Configure sign-in experience**, choose the identity **Provider types** that you want to associate with your user pool.
5. Under **Cognito user pool sign-in options**, choose any combination of **User name**, **Email**, and **Phone number**.
6. Under **User name requirements**, choose **Allow users to sign in with a preferred user name** so that your users can set an alternate username when they sign in.
7. Choose **Next**, and then complete all of the steps in the wizard.

Option 2: Email address or phone number as a sign-in attribute (username attributes)

When the user signs up with an email address or phone number as their username, you can choose if they can sign up with only email addresses, only phone numbers, or either one.

To choose username attributes, don't select **User name** as a sign-in option when you create your user pool.

The email address or phone number must be unique, and it must not already be in use by another user. It doesn't have to be verified. After the user has signed up with an email address or phone number, the user can't create a new account with the same email address or phone number. The user can only reuse the existing account and reset the account password, if needed. However, the

user can change the email address or phone number to a new email address or phone number. If the email address or phone number isn't already in use, it becomes the new username.

Note

If a user signs up with an email address as their username, they can change the username to another email address, but they can't change it to a phone number. If they sign up with a phone number, they can change the username to another phone number, but they can't change it to an email address.

Use the following steps during the user pool creation process to set up sign-up and sign-in with email address or phone number.

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

1. Go to [Amazon Cognito](#) in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. In the top-right corner of the page, choose **Create a user pool** to start the user pool creation wizard.
4. Under **Cognito user pool sign-in options**, choose any combination of **Email** and **Phone number** that represents the attributes that the user can use to sign in.
5. Choose **Next**, and then complete the remaining steps in the wizard.

Note

You do not need to mark email address 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 address 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 isn't in email address or phone number format, the `SignUp` API returns an exception.
- The `SignUp` API generates a persistent [UUID](#) for your user, and uses it internally as the immutable `username` attribute. 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 returns 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 50 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 has the following characteristics:

- You can define it as a string or a number. Amazon Cognito writes custom attribute values to the ID token only as strings.
- You can't require that users provide a value for the attribute.
- You can't remove or change it after you add it to the user pool.
- The character length of the attribute name is within the limit that Amazon Cognito accepts. For more information, see [Quotas in Amazon Cognito](#).
- It can be *mutable* or *immutable*. You can only write a value to an immutable attribute when you create a user. You can change the value of a mutable attribute if your app client has write permission to the attribute. See [Attribute permissions and scopes](#) for more information.

Note

In your code, and in rules settings for [Using role-based access control](#), custom attributes require the custom: prefix to distinguish them from standard attributes.

You can also add *developer attributes* when you create user pools, in the SchemaAttributes property of [CreateUserPool](#). Developer attributes have a dev: prefix. You can only modify a user's developer attributes with AWS credentials. Developer attributes are a legacy feature that Amazon Cognito replaced with app client read-write permissions.

Use the following procedure to create a new custom attribute.

To add a custom attribute using the console

1. Go to [Amazon Cognito](#) in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Sign-up** menu, and in the **Custom attributes** section, choose **Add custom attributes**.
5. On the **Add custom attributes** page, provide the following details about the new attribute:
 - Enter a **Name**.
 - Select a **Type** of either **String** or **Number**.
 - Enter a **Min** string length or number value.
 - Enter a **Max** string length or number value.
 - Select **Mutable** if you want to give users permission to change the value of a custom attribute after they set the initial value.
6. Choose **Save changes**.

Attribute permissions and scopes

For each app client, you can set read and write permissions for each user attribute. This way, you can control the access that any app has to read and modify each attribute that you store for your users. For example, you might have a custom attribute that indicates whether a user is a paying

customer or not. Your apps might be able to see this attribute but not change it directly. Instead, you would update this attribute using an administrative tool or a background process. You can set permissions for user attributes from the Amazon Cognito console, the Amazon Cognito API, or the AWS CLI. By default, any new custom attributes aren't available until you set read and write permissions for them. By default, when you create a new app client, you grant your app read and write permissions for all standard and custom attributes. To limit your app to only the amount of information that it requires, assign specific permissions to attributes in your app client configuration.

As a best practice, specify attribute read and write permissions when you create an app client. Grant your app client access to the minimum set of user attributes that you need for the operation of your application.

 **Note**

[DescribeUserPoolClient](#) only returns values for `ReadAttributes` and `WriteAttributes` when you configure app client permissions other than the default.

To update attribute permissions (AWS Management Console)

1. Go to [Amazon Cognito](#) in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **App clients** menu and choose an app client from the list.
5. In the **Attribute permissions** tab, choose **Edit**.
6. On the **Edit attribute read and write permissions** page, configure your read and write permissions, and then choose **Save changes**.

Repeat these steps for each app client that uses the custom attribute.

For each app client, you can mark attributes as readable or writeable. This applies to both standard and custom attributes. Your app can retrieve the value of attributes that you mark as readable, and can set or modify the value of attributes that you mark as writeable. If your app tries to set a value for an attribute that it isn't authorized to write, Amazon Cognito returns `NotAuthorizedException`. [GetUser](#) requests include an access token with an app client claim;

Amazon Cognito only returns values for attributes that your app client can read. Your user's ID token from an app only contains claims that correspond to the readable attributes. All app clients can write user pool required attributes. You can only set the value of an attribute in an Amazon Cognito user pools API request when you also provide a value for any required attributes that don't yet have a value.

Custom attributes have distinct features for read and write permissions. You can create them as mutable or immutable for the user pool, and you can set them as read or write attributes for any app client.

An immutable custom attribute can be updated once, during user creation. You can populate an immutable attribute with the following methods.

- **SignUp:** A user signs up with an app client that has write access to an immutable custom attribute. They provide a value for that attribute.
- **Sign-in with a third-party IdP:** A user signs in to an app client that has write access to an immutable custom attribute. Your user pool configuration for their IdP has a rule to map a provided claim to an immutable attribute. This is possible but not practical, because the user will only be able to sign in one time. On sign-in attempts after their first, Amazon Cognito rejects the attempt because of the mapping rule to a now-unwriteable attribute.
- **AdminCreateUser:** You provide a value for an immutable attribute.

Attribute permissions with scopes

In user pools that you configure with an AWS SDK or CDK, the REST API, or the AWS CLI, you can configure app client read or write access with the OIDC scope `oidc:profile`. The `oidc:profile` grants read or write access to the following standard attributes:

- `name`
- `family_name`
- `given_name`
- `middle_name`
- `nickname`
- `preferred_username`
- `profile`
- `picture`

- website
- gender
- birthdate
- zoneinfo
- locale

This list is the OIDC standard attributes minus email, phone_number, sub, and address, as defined in [section 2.4 of the OIDC specification](#). For information about the scopes that you can assign to your app clients, see [Scopes, M2M, and APIs with resource servers](#).

To configure your app client to write to the attributes under the oidc:profile scope, set the value of [WriteAttributes](#) to oidc:profile, plus any other attributes that you want to permit your application to modify, in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) API request. Similarly, to grant read access to these attributes, add oidc:profile to the value of [ReadAttributes](#).

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

Understanding user pool JSON web tokens (JWTs)

Tokens authenticate users and grant access to resources. The *claims* in tokens are information about your user. The ID token contains claims about their identity, like their username, family name, and email address. The access token contains claims like scope that the authenticated user can use to access third-party APIs, Amazon Cognito user self-service API operations, and the [userInfo endpoint](#). The access and ID tokens both include a cognito:groups claim that contains your user's group membership in your user pool. For more information about user pool groups, see [Adding groups to a user pool](#).

Amazon Cognito also has refresh tokens that you can use to get new tokens or revoke existing tokens. [Refresh a token](#) to retrieve a new ID and access tokens. [Revoke a token](#) to revoke user access that is allowed by refresh tokens.

Amazon Cognito issues tokens as Base64-encoded strings. You can decode any Amazon Cognito ID or access token from base64 to plaintext JSON. Amazon Cognito refresh tokens are encrypted, opaque to user pools users and administrators, and can only be read by your user pool.

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 token, an access token, and a refresh token for the authenticated user. You can use the tokens to grant your users access to downstream resources and APIs like Amazon API Gateway. Or you can exchange them for temporary AWS credentials to access other AWS services.



Storing tokens

Your app must be able to store tokens of varying sizes. Token size can change for reasons including, but not limited to, additional claims, changes in encoding algorithms, and changes in encryption algorithms. When you enable token revocation in your user pool, Amazon Cognito adds additional claims to JSON Web Tokens, increasing their size. The new claims `origin_jti` and `jti` are added to access and ID tokens. For more information about token revocation, see [Revoking tokens](#).

⚠ Important

As a best practice, secure all tokens in transit and storage in the context of your application. Tokens can contain personally-identifying information about your users, and information about the security model that you use for your user pool.

Customizing tokens

You can customize the access and ID tokens that Amazon Cognito passes to your app. In a [Pre token generation Lambda trigger](#), you can add, modify, and suppress token claims. The pre token generation trigger is a Lambda function that Amazon Cognito sends a default set of claims to. The claims include OAuth 2.0 scopes, user pool group membership, user attributes, and others. The function can then take the opportunity to make changes at runtime and return updated token claims to Amazon Cognito.

Additional costs apply to access token customization with version 2 events. For more information, see [Amazon Cognito Pricing](#).

Topics

- [Understanding the identity \(ID\) token](#)
- [Understanding the access token](#)
- [Understanding the refresh token](#)
- [Ending user sessions with token revocation](#)
- [Verifying a JSON Web Token](#)
- [Managing user pool token expiration and caching](#)

Understanding the identity (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](#).

You can set the ID token expiration to any value between 5 minutes and 1 day. You can set this value per app client.

Important

When your user signs in with managed login, Amazon Cognito sets session cookies that are valid for 1 hour. If you use managed login for authentication in your application, and specify a minimum duration of less than 1 hour for your access and ID tokens, your users will still have a valid session until the cookie expires. If the user has tokens that expire during the one-hour session, the user can refresh their tokens without the need to reauthenticate.

ID Token Header

The header contains two pieces of information: the key ID (kid), and the algorithm (alg).

```
{  
  "kid" : "1234example=",  
  "alg" : "RS256"
```

}

kid

The key ID. Its value indicates the key that was used to secure the JSON Web Signature (JWS) of the token. You can view your user pool signing key IDs at the `jwks_uri` endpoint.

For more information about the kid parameter, see the [Key identifier \(kid\) header parameter](#).

alg

The cryptographic algorithm that Amazon Cognito 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](#).

ID token default payload

This is a example payload from an ID token. It contains claims about the authenticated user. For more information about OpenID Connect (OIDC) standard claims, see the list of [OIDC standard claims](#). You can add claims of your own design with a Pre token generation Lambda trigger.

```
<header>.{  
  "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee",  
  "cognito:groups": [  
    "test-group-a",  
    "test-group-b",  
    "test-group-c"  
,  
  "email_verified": true,  
  "cognito:preferred_role": "arn:aws:iam::111122223333:role/my-test-role",  
  "iss": "https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",  
  "cognito:username": "my-test-user",  
  "middle_name": "Jane",  
  "nonce": "abcdefg",  
  "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee",  
  "cognito:roles": [  
    "arn:aws:iam::111122223333:role/my-test-role"  
,  
  "aud": "xxxxxxxxxxxxexample",  
  "identities": [  
    {
```

```
        "userId": "amzn1.account.EXAMPLE",
        "providerName": "LoginWithAmazon",
        "providerType": "LoginWithAmazon",
        "issuer": null,
        "primary": "true",
        "dateCreated": "1642699117273"
    },
],
"event_id": "64f513be-32db-42b0-b78e-b02127b4f463",
"token_use": "id",
"auth_time": 1676312777,
"exp": 1676316377,
"iat": 1676312777,
"jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee",
"email": "my-test-user@example.com"
}
.<token signature>
```

sub

A unique identifier ([UUID](#)), or subject, for the authenticated user. The username might not be unique in your user pool. The sub claim is the best way to identify a given user.

cognito:groups

An array of the names of user pool groups that have your user as a member. Groups can be an identifier that you present to your app, or they can generate a request for a preferred IAM role from an identity pool.

cognito:preferred_role

The ARN of the IAM role that you associated with your user's highest-priority user pool group. For more information about how your user pool selects this role claim, see [Assigning precedence values to groups](#).

iss

The identity provider that issued the token. The claim has the following format.

<https://cognito-idp.<Region>.amazonaws.com/<your user pool ID>>

cognito:username

The username of your user in your user pool.

nonce

The nonce claim comes from a parameter of the same name that you can add to requests to your OAuth 2.0 authorize endpoint. When you add the parameter, the nonce claim is included in the ID token that Amazon Cognito issues, and you can use it to guard against replay attacks. If you do not provide a nonce value in your request, Amazon Cognito automatically generates and validates a nonce when you authenticate through a third-party identity provider, then adds it as a nonce claim to the ID token. The implementation of the nonce claim in Amazon Cognito is based on [OIDC standards](#).

origin_jti

A token-revocation identifier associated with your user's refresh token. Amazon Cognito references the `origin_jti` claim when it checks if you revoked your user's token with the [Revoke endpoint](#) or the [RevokeToken](#) API operation. When you revoke a token, Amazon Cognito invalidates all access and ID tokens with the same `origin_jti` value.

cognito:roles

An array of the names of the IAM roles associated with your user's groups. Every user pool group can have one IAM role associated with it. This array represents all IAM roles for your user's groups, regardless of precedence. For more information, see [Adding groups to a user pool](#).

aud

The user pool app client that authenticated your user. Amazon Cognito renders the same value in the access token `client_id` claim.

identities

The contents of the user's `identities` attribute. The attribute contains information about each third-party identity provider profile that you've linked to a user, either by federated sign-in or by [linking a federated user to a local profile](#). This information contains their provider name, their provider unique ID, and other metadata.

token_use

The intended purpose of the token. In an ID token, its value is `id`.

auth_time

The authentication time, in Unix time format, that your user completed authentication.

exp

The expiration time, in Unix time format, that your user's token expires.

iat

The issued-at time, in Unix time format, that Amazon Cognito issued your user's token.

jti

The unique identifier of the JWT.

The ID token can contain OIDC standard claims that are defined in [OIDC standard claims](#). The ID token can also contain custom attributes that you define in your user pool. Amazon Cognito writes custom attribute values to the ID token as strings regardless of attribute type.

Note

User pool custom attributes are always prefixed with `custom:`.

ID Token Signature

The signature of the ID token is calculated based on the header and payload of the JWT token. Before you accept the claims in any ID token that your app receives, verify the signature of the token. For more information, see [Verifying a JSON Web Token](#). [Verifying a JSON Web Token](#).

Understanding 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. Your user pool accepts access tokens to authorize user self-service operations. For example, you can use the access token to grant your user access to add, change, or delete user attributes.

With [OAuth 2.0 scopes](#) in an access token, derived from the custom scopes that you add to your user pool, you can authorize your user to retrieve information from an API. For example, Amazon API Gateway supports authorization with Amazon Cognito access tokens. You can populate a REST API authorizer with information from your user pool, or use Amazon Cognito as a JSON Web Token (JWT) authorizer for an HTTP API. To generate an access token with custom scopes, you must request it through your user pool [public endpoints](#).

Your user's access token is permission to request more information about your user's attributes from the [userInfo endpoint](#). Your user's access token is also permission to read and write user

attributes. The level of access to attributes that your access token grants depends on the permissions you assign to your app client, and the scopes that you grant in the token.

The access token is a [JSON Web Token \(JWT\)](#). The header for the access token has the same structure as the ID token. Amazon Cognito signs access tokens with a different key from the key that signs ID tokens. The value of an access key ID (kid) claim won't match the value of the kid claim in an ID token from the same user session. In your app code, verify ID tokens and access tokens independently. Don't trust the claims in an access token until you verify the signature. For more information, see [Verifying a JSON Web Token](#). You can set the access token expiration to any value between 5 minutes and 1 day. You can set this value per app client.

Important

For access and ID tokens, don't specify a minimum less than an hour if you use managed login. 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",  
}
```

kid

The key ID. Its value indicates the key that was used to secure the JSON Web Signature (JWS) of the token. You can view your user pool signing key IDs at the `jwks_uri` endpoint.

For more information about the `kid` parameter, see the [Key identifier \(kid\) header parameter](#).

alg

The cryptographic algorithm that Amazon Cognito 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 default payload

This is a sample payload from an access token. For more information, see [JWT claims](#). You can add claims of your own design with a [Pre token generation Lambda trigger](#).

```
<header>.  
{  
    "sub": "aaaaaaaa-bbbbcccc-dddd-eeeeeeeeeee",  
    "device_key": "aaaaaaaa-bbbbcccc-dddd-eeeeeeeeeee",  
    "cognito:groups": [  
        "testgroup"  
    ],  
    "iss": "https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",  
    "version": 2,  
    "client_id": "xxxxxxxxxxxxexample",  
    "origin_jti": "aaaaaaaa-bbbbcccc-dddd-eeeeeeeeeee",  
    "event_id": "aaaaaaaa-bbbbcccc-dddd-eeeeeeeeeee",  
    "token_use": "access",  
    "scope": "phone openid profile resourcesserver.1/appclient2 email",  
    "auth_time": 1676313851,  
    "exp": 1676317451,  
    "iat": 1676313851,  
    "jti": "aaaaaaaa-bbbbcccc-dddd-eeeeeeeeeee",  
    "username": "my-test-user"  
}  
.<token signature>
```

sub

A unique identifier ([UUID](#)), or subject, for the authenticated user. The username might not be unique in your user pool. The sub claim is the best way to identify a given user.

cognito:groups

An array of the names of user pool groups that have your user as a member.

iss

The identity provider that issued the token. The claim has the following format.

`https://cognito-idp.<Region>.amazonaws.com/<your user pool ID>`

client_id

The user pool app client that authenticated your user. Amazon Cognito renders the same value in the ID token aud claim.

origin_jti

A token-revocation identifier associated with your user's refresh token. Amazon Cognito references the origin_jti claim when it checks if you revoked your user's token with the [Revoke endpoint](#) or the [RevokeToken](#) API operation. When you revoke a token, Amazon Cognito invalidates all access and ID tokens with the same origin_jti value.

token_use

The intended purpose of the token. In an access token, its value is access.

scope

A list of OAuth 2.0 scopes that define what access the token provides. A token from the [Token endpoint](#) can contain any scopes that your app client supports. A token from Amazon Cognito API sign-in only contains the scope aws.cognito.signin.user.admin.

auth_time

The authentication time, in Unix time format, that your user completed authentication.

exp

The expiration time, in Unix time format, that your user's token expires.

iat

The issued-at time, in Unix time format, that Amazon Cognito issued your user's token.

jti

The unique identifier of the JWT.

username

The username of your user in your user pool.

More resources

- [How to customize access tokens in Amazon Cognito user pools](#)

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 [Verifying a JSON Web Token](#).

Understanding 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 re-authenticate 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 be continually used to retrieve new access and ID tokens. You will see expected behavior with a minimum of 7 minutes instead of 5 minutes.

Your user's account itself doesn't expire, as long as the user has logged in at least once before the UnusedAccountValidityDays time limit for new accounts.

Getting new access and identity tokens with a refresh token

Use the API or managed login to initiate authentication for refresh tokens.

To use the refresh token to get new ID and access tokens with the user pools API, use the [AdminInitiateAuth](#) or [InitiateAuth](#) API operations. Pass REFRESH_TOKEN_AUTH for the AuthFlow parameter. In the AuthParameters property of AuthFlow, pass your user's refresh token as the value of "REFRESH_TOKEN". Amazon Cognito returns new ID and access tokens after your API request passes all challenges.

Note

To use the Amazon Cognito user pools API to refresh tokens for a managed login user, generate an `InitiateAuth` request with the `REFRESH_TOKEN_AUTH` flow. This method of token handling in your application doesn't affect users' managed login sessions. The API

response issues new ID and access tokens, but doesn't renew the managed login session cookie.

You can also submit refresh tokens to the [Token endpoint](#) in a user pool where you have configured a domain. In the request body, include a grant_type value of refresh_token and a refresh_token value of your user's refresh token.

Revoking refresh tokens

You can revoke refresh tokens that belong to a user. For more information about revoking tokens, see [Ending user sessions with token revocation](#).

Note

Revoking the refresh token will revoke all ID and access tokens that Amazon Cognito issued from refresh requests with that token.

Users can sign out from all devices where they are currently signed in when you revoke all of the user's tokens using the GlobalSignOut and AdminUserGlobalSignOut API operations. After the user is signed out, the following effects happen.

- The user's refresh token can't get new tokens for the user.
- The user's access token can't make token-authorized API requests.
- The user must re-authenticate to get new tokens. Because managed login session cookies don't expire automatically, your user can re-authenticate with a session cookie, with no additional prompt for credentials. After you sign out your managed login users, redirect them to the [Logout endpoint](#), where Amazon Cognito will clear their session cookie.

With refresh tokens, you can persist users' sessions in your app for a long time. Over time, your users might want to deauthorize some devices where they have signed in, continually refreshing their session. To sign your user out from a single device, revoke their refresh token. When your user wants to sign themselves out from all authenticated sessions, generate a [GlobalSignOut](#) API request . Your app can present your user with a choice like **Sign out from all devices**. GlobalSignOut accepts a user's valid–unaltered, unexpired, not-revoked–access token. Because this API is token-authorized, one user can't use it to initiate sign-out for another user.

You can, however, generate an [AdminUserGlobalSignOut](#) API request that you authorize with your AWS credentials to sign out any user from all of their devices. The administrator application must call this API operation with AWS developer credentials and pass the user pool ID and the user's username as parameters. The AdminUserGlobalSignOut API can sign out any user in the user pool.

For more information about requests that you can authorize with either AWS credentials or a user's access token, see [Amazon Cognito user pools authenticated and unauthenticated API operations](#).

Ending user sessions with token revocation

You can revoke a refresh token for a user using the user pools API or the authorization server [Revoke endpoint](#). 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 Amazon 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 a 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-idp describe-user-pool-client` CLI command or the `DescribeUserPoolClient` API operation to retrieve the current settings from your app client. Then call the `aws cognito-idp update-user-pool-client` CLI command or the `UpdateUserPoolClient` API operation. Include the current settings from your app client and 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.

To create or modify an app client with token revocation enabled, include the following parameter in your [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) API request.

```
"EnableTokenRevocation": true
```

Revoke a token

You can revoke a refresh token using a [RevokeToken](#) API request, for example with the [aws cognito-idp revoke-token](#) CLI command. You can also revoke tokens using the [Revoke endpoint](#). This endpoint is available after you add a domain to your user pool. You can use the revocation endpoint on either an Amazon Cognito hosted domain or your own custom domain.

Note

Your request to revoke a refresh token must include the client ID that was used to obtain the token.

The following is the body of an example RevokeToken API request.

```
{
  "ClientId": "1example23456789",
  "ClientSecret": "abcdef123456789ghijklexample",
  "Token": "eyJjdHkiOiJKV1QiEXAMPLE"
}
```

The following is an example cURL request to the /oauth2/revoke endpoint of a user pool with a custom domain.

```
curl --location 'auth.mydomain.com/oauth2/revoke' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Authorization: Basic Base64Encode(client_id:client_secret)' \
--data-urlencode 'token=abcdef123456789ghijklexample' \
--data-urlencode 'client_id=1example23456789'
```

The RevokeToken operation and the /oauth2/revoke endpoint require no additional authorization unless your app client has a client secret.

Verifying a JSON Web Token

JSON web tokens (JWTs) can be decoded, read, and modified easily. A modified access token creates a risk of privilege escalation. A modified ID token creates a risk of impersonation. Your application trusts your user pool as a token issuer, but what if a user intercepts the token in transit? You must ensure that your application is receiving the same token that Amazon Cognito issued.

Amazon Cognito issues tokens that use some of the integrity and confidentiality features of the OpenID Connect (OIDC) specification. User pool tokens indicate validity with objects like the expiration time, issuer, and digital signature. The signature, the third and final segment of the .-delimited JWT, is the key component of token validation. A malicious user can modify a token, but if your application retrieves the public key and compares the signature, it won't match. Any application that processes JWTs from OIDC authentication must perform this verification operation with each sign-in.

On this page, we make some general and specific recommendations for verification of JWTs. Application development spans a variety of programming languages and platforms. Because Amazon Cognito implements OIDC sufficiently close to the public specification, any reputable JWT library in your developer environment of choice can handle your verification requirements.

These steps describe verifying a user pool JSON Web Token (JWT).

Topics

- [Prerequisites](#)
- [Validate tokens with aws-jwt-verify](#)
- [Understanding and inspecting tokens](#)

Prerequisites

Your library, SDK, or software framework might already handle the tasks in this section. AWS SDKs provide tools for Amazon Cognito user pool token handling and management in your app. AWS Amplify includes functions to retrieve and refresh Amazon Cognito tokens.

For more information, see the following pages.

- [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#)

- [Code examples for Amazon Cognito Identity Provider using AWS SDKs](#)
- [Advanced workflows in the Amplify Dev Center](#)

Many libraries are available for decoding and verifying a JSON Web Token (JWT). If you want to manually process tokens for server-side API processing, or if you are using other programming languages, these libraries can help. See the [OpenID foundation list of libraries for working with JWT tokens](#).

Validate tokens with aws-jwt-verify

In a Node.js app, AWS recommends the [aws-jwt-verify library](#) to validate the parameters in the token that your user passes to your app. With aws-jwt-verify, you can populate a `CognitoJwtVerifier` with the claim values that you want to verify for one or more user pools. Some of the values that it can check include the following.

- That access or ID tokens aren't malformed or expired, and have a valid signature.
- That access tokens came from the [correct user pools and app clients](#).
- That access token claims contain the [correct OAuth 2.0 scopes](#).
- That the keys that signed your access and ID tokens [match a signing key kid from the JWKS URI of your user pools](#).

The JWKS URI contains public information about the private key that signed your user's token. You can find the JWKS URI for your user pool at [https://cognito-idp.<Region>.amazonaws.com/<userPoolId>/well-known/jwks.json](https://cognito-idp.<Region>.amazonaws.com/<userPoolId>/.well-known/jwks.json).

For more information and example code that you can use in a Node.js app or a AWS Lambda authorizer, see [aws-jwt-verify](#) on GitHub.

Understanding and inspecting tokens

Before you integrate token inspection with your app, consider how Amazon Cognito assembles JWTs. Retrieve example tokens from your user pool. Decode and examine them in detail to understand their characteristics, and determine what you want to verify and when. For example, you might want to examine group membership in one scenario, and scopes in another.

The following sections describe a process to manually inspect Amazon Cognito JWTs as you prepare your app.

Confirm the structure of the JWT

A JSON Web Token (JWT) includes three sections with a . (dot) delimiter between them.

Header

The key ID, kid, and the RSA algorithm, alg, that Amazon Cognito used to sign the token. Amazon Cognito signs tokens with an alg of RS256. The kid is a truncated reference to a 2048-bit RSA private signing key held by your user pool.

Payload

Token claims. In an ID token, the claims include user attributes and information about the user pool, iss, and app client, aud. In an access token, the payload includes scopes, group membership, your user pool as iss, and your app client as client_id.

Signature

The signature isn't decodable base64 like the header and payload. It's an RSA256 identifier derived from a signing key and parameters that you can observe at your JWKS URI.

The header and payload are base64-encoded JSON. You can identify them by the opening characters eyJ that decode to the starting character {. If your user presents a base64-encoded JWT to your app and it's not in the format [JSON Header].[JSON Payload].[Signature], it's not a valid Amazon Cognito token and you can discard it.

Validate the JWT

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 private key signs access tokens, and the other signs ID tokens.

To verify the signature of a JWT token

1. Decode the ID token.

The OpenID Foundation also [maintains a list of libraries for working with JWT tokens](#).

You can also use AWS Lambda to decode user pool JWTs. For more information, see [Decode and verify Amazon Cognito JWT tokens using AWS Lambda](#).

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 by constructing the following jwks_uri URI for your environment:

```
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**

Amazon Cognito might rotate signing keys in your user pool. As a best practice, cache public keys in your app, using the kid as a cache key, and refresh the cache periodically. Compare the kid in the tokens that your app receives to your cache. If you receive a token with the correct issuer but a different kid, Amazon Cognito might have rotated the signing key. Refresh the cache from your user pool jwks_uri endpoint.

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 a JWT library to compare the signature of the issuer to the signature in the token. The issuer signature is derived from the public key (the RSA modulus "`n`") of the `kid` in `jwks.json` that matches the token `kid`. You might need to convert the JWK to PEM format first. The following 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) {
});
```

Verify the claims

To verify JWT claims

1. By one of the following methods, verify that the token hasn't expired.
 - a. Decode the token and compare the `exp` claim to the current time.
 - b. If your access token includes an `aws.cognito.signin.user.admin` claim, send a request to an API like [GetUser](#). API requests that you [authorize with an access token](#) return an error if your token has expired.
 - c. Present your access token in a request to the [userInfo endpoint](#). Your request returns an error if your token has expired.
2. The `aud` claim in an ID token and the `client_id` claim in an access token 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.

Managing user pool token expiration and caching

Your app must successfully complete one of the following requests each time you want to get a new JSON Web Token (JWT).

- Request a client credentials or authorization code [grant](#) from the [Token endpoint](#).
- Request an implicit grant from your managed login pages.
- Authenticate a local user in an Amazon Cognito API request like [InitiateAuth](#).

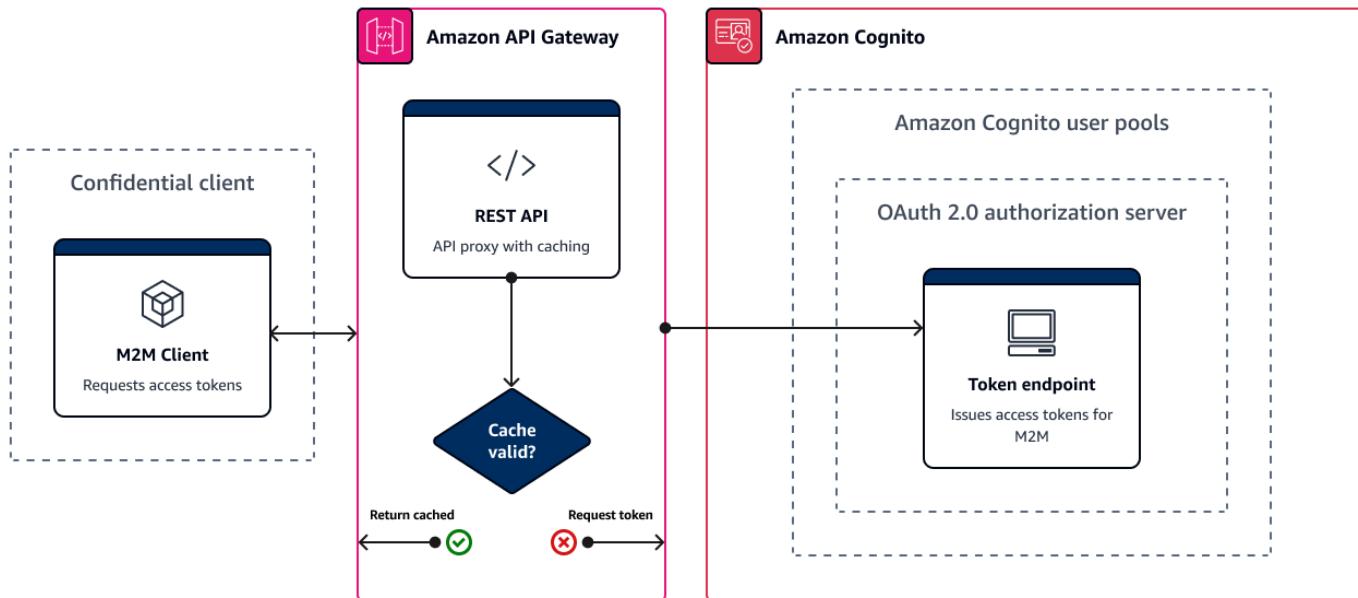
You can configure your user pool to set tokens to expire in minutes, hours, or days. To ensure the performance and availability of your app, use Amazon Cognito tokens for about 75% of the token lifetime, and only then retrieve new tokens. A cache solution that you build for your app keeps tokens available, and prevents the rejection of requests by Amazon Cognito when your request rate is too high. A client-side app must store tokens in a memory cache. A server-side app can add an encrypted cache mechanism to store tokens.

When your user pool generates a high volume of user or machine-to-machine activity, you might encounter the limits that Amazon Cognito sets on the number of requests for tokens that you can make. To reduce the number of requests you make to Amazon Cognito endpoints, you can either securely store and reuse authentication data, or implement exponential backoff and retries.

Authentication data comes from two classes of endpoints. Amazon Cognito [OAuth 2.0 endpoints](#) include the token endpoint, which services client credentials and managed login authorization code requests. [Service endpoints](#) answer user pools API requests like `InitiateAuth` and `RespondToAuthChallenge`. Each type of request has its own limit. For more information about limits, see [Quotas in Amazon Cognito](#).

Caching machine-to-machine access tokens with Amazon API Gateway

With API Gateway token caching, your app can scale in response to events larger than the default request rate quota of Amazon Cognito OAuth endpoints.



You can cache the access tokens so that your app only requests a new access token if a cached token is expired. Otherwise, your caching endpoint returns a token from the cache. This prevents an additional call to an Amazon Cognito API endpoint. When you use Amazon API Gateway as a proxy to the [Token endpoint](#), your API responds to the majority of requests that would otherwise contribute to your request quota, avoiding unsuccessful requests as a result of rate limiting.

The following API Gateway-based solution offers a low-latency, low-code/no-code implementation of token caching. API Gateway APIs are encrypted in transit, and optionally at rest. An API Gateway cache is ideal for the OAuth 2.0 [client credentials grant](#), a frequently high-volume grant type that produces access tokens to authorize machine-to-machine and microservice sessions. In an event like a traffic surge that causes your microservices to horizontally scale, you can end up with many systems using the same client credentials at a volume that exceeds the AWS request-rate limit of your user pool or app client. To preserve app availability and low latency, a caching solution is best practice in such scenarios.

In this solution, you define a cache in your API to store a separate access token for each combination of OAuth scopes and app client that you want to request in your app. When your app makes a request that matches the cache key, your API responds with an access token that Amazon Cognito issued to the first request that matched the cache key. When your cache key duration expires, your API forwards the request to your token endpoint and caches a new access token.

Note

Your cache key duration must be shorter than the access token duration of your app client.

The cache key is a combination of the OAuth scopes that you request in the scope URL parameter and the Authorization header in the request. The Authorization header contains your app client ID and client secret. You don't need to implement additional logic in your app to implement this solution. You must only update your configuration to change the path to your user pool token endpoint.

You can also implement token caching with [ElastiCache \(Redis OSS\)](#). For fine-grained control with AWS Identity and Access Management (IAM) policies, consider an [Amazon DynamoDB](#) cache.

Note

Caching in API Gateway is subject to additional cost. [See pricing for more details.](#)

To set up a caching proxy with API Gateway

1. Open the [API Gateway console](#) and create a REST API.
2. In **Resources**, create a POST method.
 - a. Choose the **HTTP Integration type**.
 - b. Select **Use HTTP proxy integration**.
 - c. Enter an **Endpoint URL** of `https://<your user pool domain>/oauth2/token`.
3. In **Resources**, configure the cache key.
 - a. Edit the **Method request** of your POST method.
 - b. Set your scope parameter and Authorization header as your caching key.
 - i. Add a query string to **URL query string parameters** and choose **Caching** for the scope string.
 - ii. Add a header to **HTTP request headers** and choose **Caching** for the Authorization header.
4. In **Stages**, configure caching.

- a. Choose the stage that you want to modify.
 - b. Under **Settings**, select **Enable API cache**.
 - c. Choose a **Cache capacity**.
 - d. Choose a **Cache time-to-live (TTL)** of 3600 seconds.
 - e. Clear the **Require authorization** check box.
5. In **Stages**, note the **Invoke URL**.
 6. Update your app to POST token requests to the **Invoke URL** of your API instead of the /oauth2/token endpoint of your user pool.

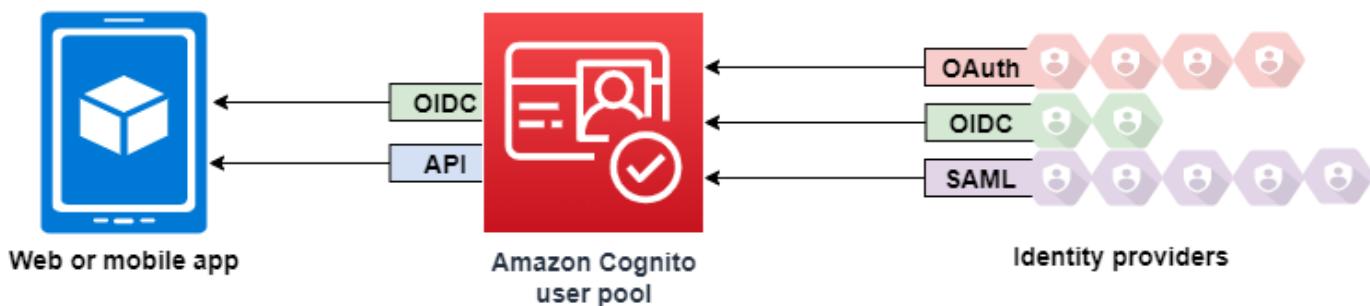
Accessing resources after successful sign-in

Your app users can either sign in directly through a user pool, or they can 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 [Understanding user pool JSON web tokens \(JWTs\)](#).

After a successful authentication, your app will receive user pool tokens from Amazon Cognito. You can use user pool tokens to:

- Retrieve AWS credentials that authorize requests for application resources in AWS services like Amazon DynamoDB and Amazon S3.
- Provide temporary, revocable proof of authentication.
- Populate identity data to a user profile in your app.
- Authorize changes to the signed-in user's profile in the user pool directory.
- Authorize requests for user information with an access token.
- Authorize requests to data that is behind access-protected external APIs with access tokens.
- Authorize access to application assets that are stored on the client or server with Amazon Verified Permissions.

For more information, see [An example authentication session](#) and [Understanding user pool JSON web tokens \(JWTs\)](#).



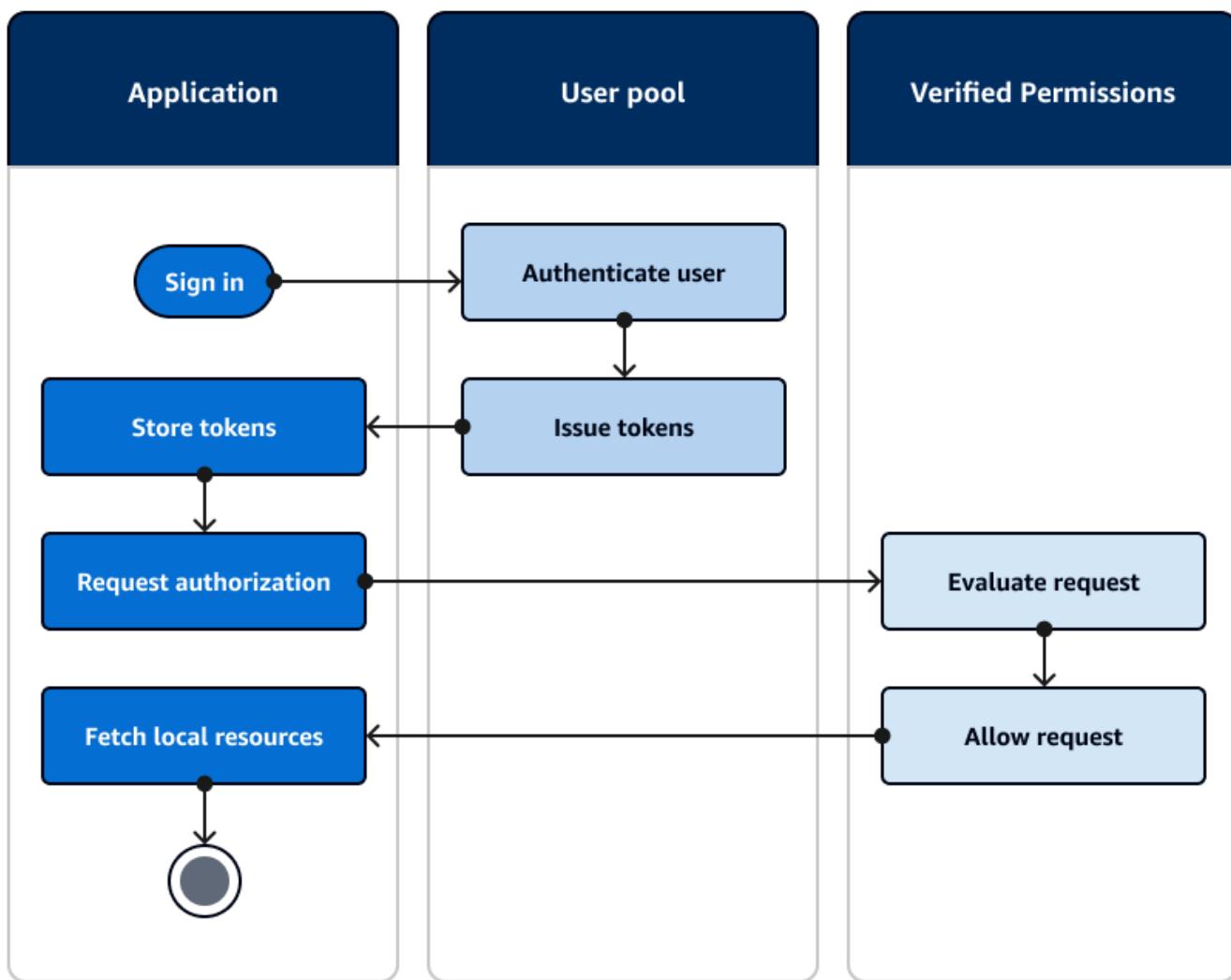
Topics

- [Authorizing access to client or server resources with Amazon Verified Permissions](#)
- [Accessing resources with API Gateway after sign-in](#)
- [Accessing AWS services using an identity pool after sign-in](#)

Authorizing access to client or server resources with Amazon Verified Permissions

Your app can pass the tokens from a signed-in user to [Amazon Verified Permissions](#). Verified Permissions is a scalable, fine-grained permissions management and authorization service for applications that you've built. An Amazon Cognito user pool can be an identity source to a Verified Permissions policy store. Verified Permissions makes authorization decisions for requested actions and resources, like GetPhoto for premium_badge.png, from the principal and their attributes in user pool tokens.

The following diagram shows how your application can pass a user's token to Verified Permissions in an authorization request.



Get started with Amazon Verified Permissions

After you integrate your user pool with Verified Permissions, you gain a central source of granular authorization for all of your Amazon Cognito apps. This removes the need for fine-grained security logic that you would otherwise have to code and replicate between all of your apps. For more information about authorization with Verified Permissions, see [Authorization with Amazon Verified Permissions](#).

Verified Permissions authorization requests require AWS credentials. You can implement some of the following techniques to safely apply credentials to authorization requests.

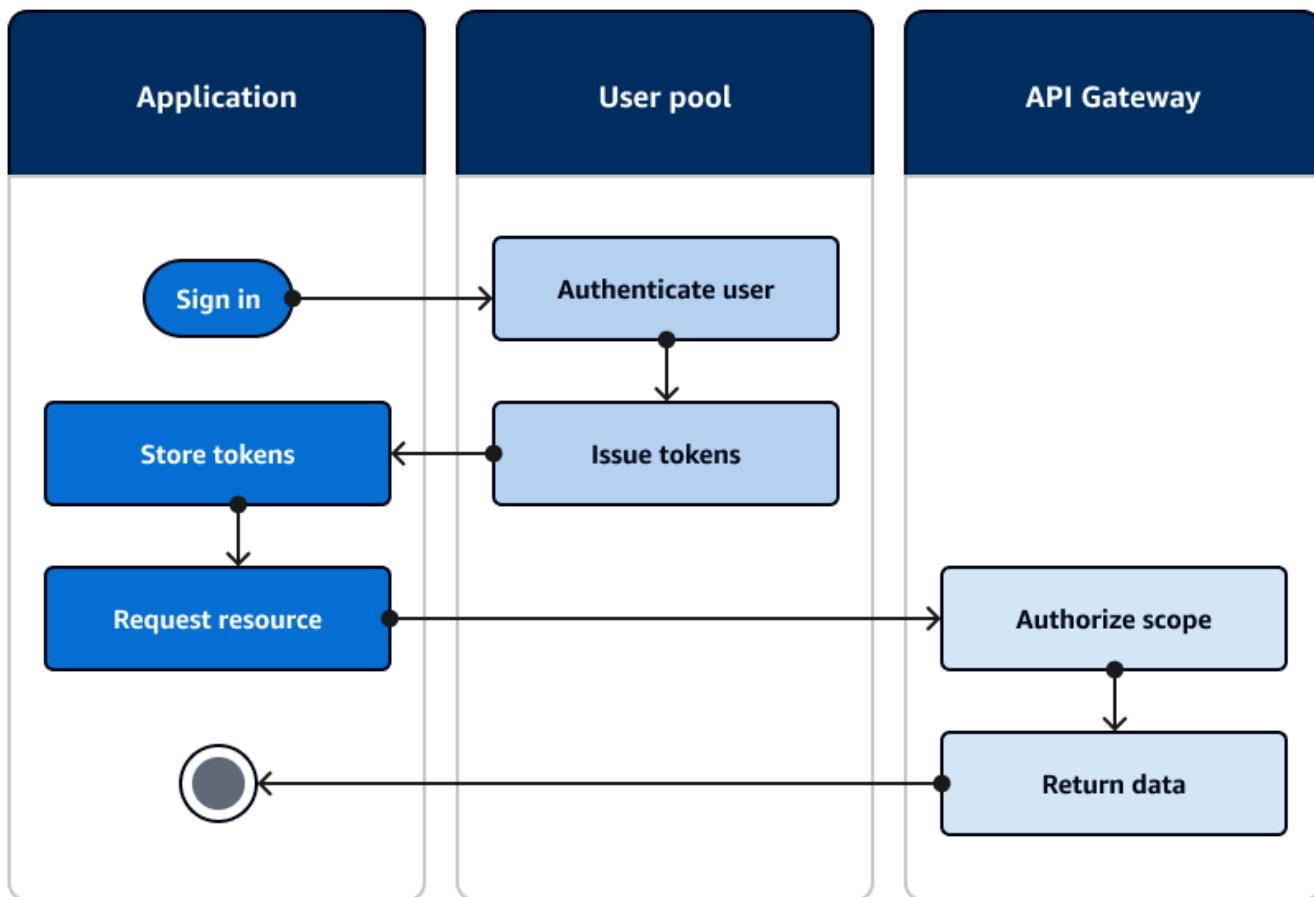
- Operate a web application that can store secrets in the server backend.

- Acquire authenticated identity pool credentials.
- Proxy user requests through an access-token-authorized API, and append AWS credentials to the request.

Accessing resources with API Gateway after sign-in

A common use of Amazon Cognito user pools tokens is to authorize requests to an [API Gateway REST API](#). The OAuth 2.0 scopes in access tokens can authorize a method and path, like HTTP GET for /app_assets. ID tokens can serve as generic authentication to an API and can pass user attributes to the backend service. API Gateway has additional custom authorization options like [JWT authorizers for HTTP APIs](#) and [Lambda authorizers](#) that can apply more fine-grained logic.

The following diagram illustrates an application that is gaining access to a REST API with the OAuth 2.0 scopes in an access token.



Your app must collect the tokens from authenticated sessions and add them as bearer tokens to an Authorization header in the request. Configure the authorizer that you configured for the API, path, and method to evaluate token contents. API Gateway returns data only if the request matches the conditions that you set up for your authorizer.

Some potential ways that API Gateway API can approve access from an application are:

- The access token is valid, isn't expired, and contains the correct OAuth 2.0 scope. The [Amazon Cognito user pools authorizer for a REST API](#) is a common implementation with a low barrier to entry. You can also evaluate the body, query string parameters, and headers of a request to this type of authorizer.
- The ID token is valid and isn't expired. When you pass an ID token to an Amazon Cognito authorizer, you can perform additional validation of the ID token contents on your application server.
- A group, claim, attribute, or role in an access or ID token meets the requirements that you define in a Lambda function. A [Lambda authorizer](#) parses the token in the request header and evaluates it for an authorization decision. You can construct custom logic in your function or make an API request to [Amazon Verified Permissions](#).

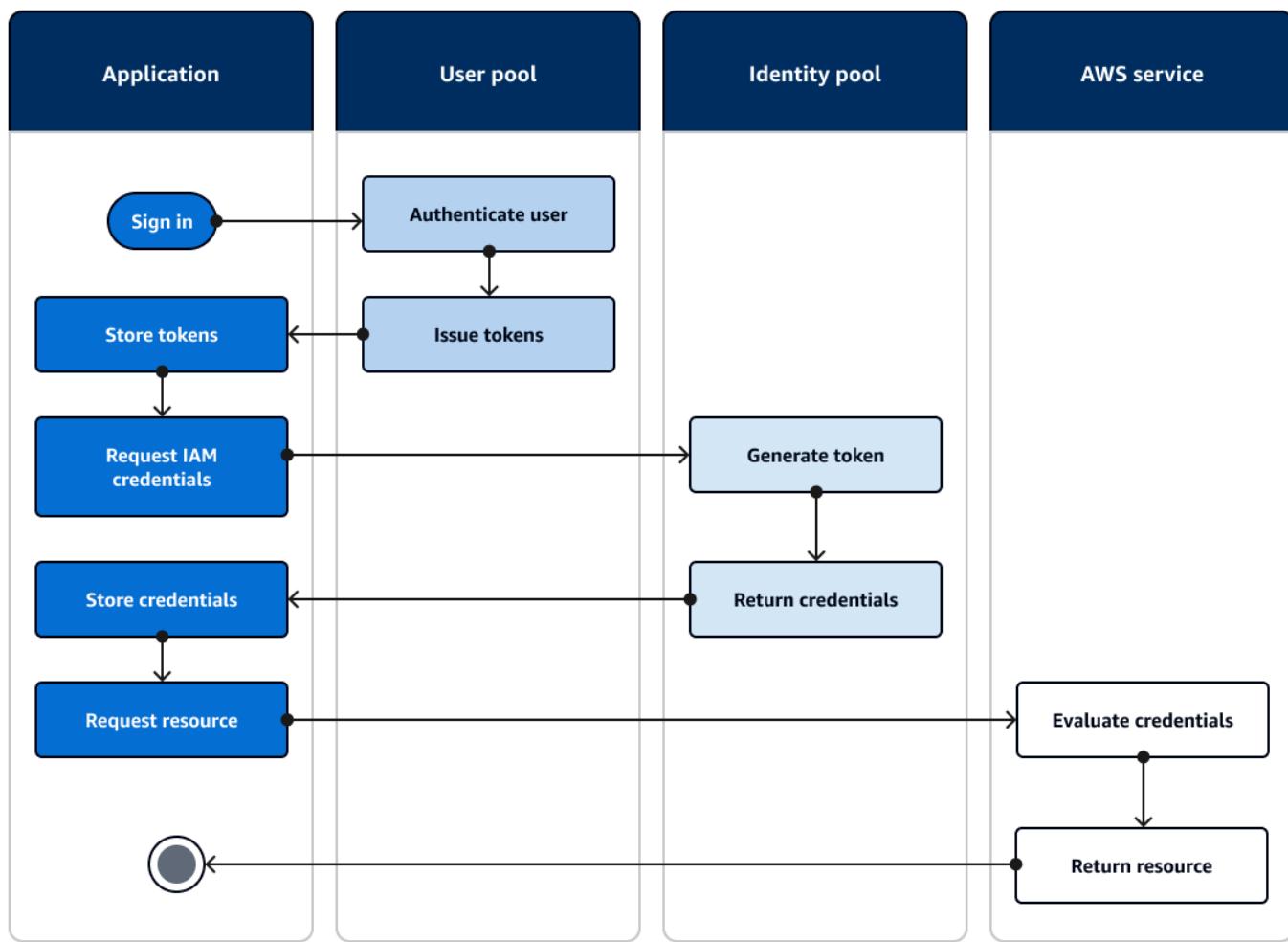
You can also authorize requests to an [AWS AppSync GraphQL API](#) with tokens from a user pool.

Accessing AWS services using an identity pool after sign-in

After your users sign in with a user pool, they can access AWS services with temporary API credentials that are issued from an identity pool.

Your web or mobile app receives tokens from a user pool. When you configure your user pool as an identity provider to your identity pool, the identity pool exchanges tokens for temporary AWS credentials. These credentials can be scoped to IAM roles and their policies that give users access to a limited set of AWS resources. For more information, see [Identity pools authentication flow](#).

The following diagram shows how an application signs in with a user pool, retrieves identity pool credentials, and requests an asset from an AWS service.



You can use identity pool credentials to:

- Make fine-grained authorization requests to Amazon Verified Permissions with your user's own credentials.
- Connect to an Amazon API Gateway REST API or an AWS AppSync GraphQL API that authorizes connections with IAM.
- Connect to a database backend like Amazon DynamoDB or Amazon RDS that authorizes connections with IAM.
- Retrieve application assets from an Amazon S3 bucket.
- Initiate a session with an Amazon WorkSpaces virtual desktop.

Identity pools don't operate exclusively within an authenticated session with a user pool. They also accept authentication directly from third-party identity providers and can generate credentials for unauthenticated guest users.

For more information about using identity pools together with user pool groups to control access to your AWS resources, see [Adding groups to a user pool](#) and [Using role-based access control](#). Also, for more information about identity pools and AWS Identity and Access Management, see [Identity pools authentication flow](#).

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](#).

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 add an Amazon Cognito user pools identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Amazon Cognito user pool**.
5. Enter a **User pool ID** and an **App client ID**.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - a. You can give users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With an Amazon Cognito user pool IdP, you can also **Choose role with preferred_role claim in tokens**. For more information about the cognito:preferred_role claim, see [Assigning precedence values to groups](#).
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to use to compare the claim to the rule,

- the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
- ii. If you chose **Choose role with preferred_role claim in tokens**, Amazon Cognito issues credentials for the role in your user's cognito:preferred_role claim. If no preferred role claim is present, Amazon Cognito issues credentials based on your **Role resolution**.
 - b. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
- To apply no principal tags, choose **Inactive**.
 - To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

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>
```

You can derive the value for **<region>** from the **User Pool ID**. For example, if the user pool ID is us-east-1_EXAMPLE1, then the **<region>** is us-east-1. If the user pool ID is us-west-2_EXAMPLE2, then the **<region>** is us-west-2.

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

```

AWSConfiguration *serviceConfiguration = [[AWSConfiguration 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, forKey: "UserPool")
let pool = AWSCognitoIdentityUserPool(forKey: "UserPool")
let credentialsProvider = AWSCognitoCredentialsProvider(regionType: .USEast1, identityPoolId: "YOUR_IDENTITY_POOL_ID", identityProviderManager:pool)
```

Configure user pool features

In previous chapters, you've likely configured some features with guidance from the Amazon Cognito console. The pages in this section are a deeper dive into the detailed configuration requirements of some of the core features of user pools. There's important reference information about your options with app clients, email and SMS configuration, remembering user devices, and more.

Topics

- [Updating user pool and app client configuration](#)
- [Application-specific settings with app clients](#)
- [Working with user devices in your user pool](#)
- [Scopes, M2M, and APIs with resource servers](#)
- [Using Amazon Pinpoint for user pool analytics](#)
- [Email settings for Amazon Cognito user pools](#)
- [SMS message settings for Amazon Cognito user pools](#)

Updating user pool and app client configuration

When you want to change a setting in a user pool or app client, you can apply the update in the Amazon Cognito console with a few clicks. You navigate through the feature-based tabs in your user pool settings and update fields as described in other areas of this guide.

Many organizations manage their resources programmatically in AWS CloudFormation, applications built on the AWS SDKs or CDK, and other automation software. When this is your resource-management model, you must take extra care when you stage changes to your resources.

The API operations [UpdateUserPool](#) and [UpdateUserPoolClient](#) make updates to an existing user pool or app client. Each comes with a warning in the API Reference: *If you don't provide a value for an attribute, Amazon Cognito sets it to its default value.* When you submit an update request with just one parameter, Amazon Cognito sets that parameter to the value of your choosing and sets all others to a default value. This can reset configurations including your attribute schema, your Lambda triggers, and your email and SMS message configuration.

Additionally, some settings are locked in after you create your user pool or app client, and you can't change them unless you create a new resource.

Topics

- [Settings you can't change](#)
- [SMS configuration](#)
- [Updating a user pool with an AWS SDK, AWS CDK, or REST API](#)

Settings you can't change

You can't change some settings after you create a user pool. If you want to change the following settings, you must create a new user pool or app client.

User pool name

API parameter name: [PoolName](#)

The friendly name that you assigned to your user pool. To change the name of a user pool, create a new user pool.

Amazon Cognito user pool sign-in options

API parameter names: [AliasAttributes](#) and [UsernameAttributes](#)

The attributes that your users can pass as a user name when they sign in. When you create a user pool, you can choose to allow sign-in with user name, email address, phone number, or a preferred user name. To change user pool sign-in options, create a new user pool.

Make user name case sensitive

API parameter name: [UsernameConfiguration](#)

When you create a user name that matches another user name except for the letter case, Amazon Cognito can treat them as either the same user or as unique users. For more information, see [User pool case sensitivity](#). To change case sensitivity, create a new user pool.

Client secret

API parameter name: [GenerateSecret](#)

When you create an app client, you can generate a client secret so that only trusted sources can make requests to your user pool. For more information, see [Application-specific settings with app clients](#). To change a client secret, create a new app client in the same user pool.

Required attributes

API parameter name: [Schema](#)

The attributes that your users must provide values for when they sign up, or when you create them. For more information, see [Working with user attributes](#). To change required attributes, create a new user pool.

Custom attributes

API parameter name: [Schema](#)

Attributes with custom names. You can change the value of a user's custom attribute, but you can't delete a custom attribute from your user pool. For more information, see [Working with user attributes](#). If you reach the maximum number of custom attributes and you want to modify the list, create a new user pool.

SMS configuration

After you activate SMS messages in your user pool, you can't deactivate them.

- If you choose to configure SMS messages when you create a user pool, you can't deactivate SMS after you complete setup.

- You can activate SMS messages in a user pool that you created, but after that you can't deactivate SMS.
- Amazon Cognito can use SMS messages for user account invitation and recovery, attribute verification, and multi-factor authentication (MFA). After you activate SMS messages, you can turn SMS messages on or off for these functions at any time.
- SMS message configuration includes an IAM role that you delegate to Amazon Cognito to send messages with Amazon SNS. You can change the assigned role at any time.

Updating a user pool with an AWS SDK, AWS CDK, or REST API

In the Amazon Cognito console, you can change your user pool settings one parameter at a time. For example, to add a Lambda trigger, you choose **Add Lambda trigger** and choose the function and trigger type. The Amazon Cognito user pools API is structured in a way that update operations for user pools and app clients require the full set of parameters for the user pool. However, the console transparently automates this update operation with your other user pool settings.

You might find at times that a change elsewhere in your AWS account can cause updates to generate an error when they aren't related to the setting you want to change. A deleted Amazon SES identity or a change in an IAM permission for AWS WAF, for example. If one of the current parameters is no longer valid, you can't update your settings until you fix it. When you encounter such an error, examine the error response and validate the setting that it mentions.

The [AWS Cloud Development Kit \(AWS CDK\)](#), [Amazon Cognito user pools REST API](#) and [AWS SDKs](#) are tools for automation and programmatic configuration of Amazon Cognito resources. Requests with these tools must also, like the Amazon Cognito console, update a setting with a full resource configuration in the request body. At a high level, you must perform the following process.

1. Capture the output from an operation that describes the configuration of your existing resource .
2. Modify the output with your settings change.
3. Send the modified configuration in an operation that updates your resource.

The following procedure updates your configuration with the [UpdateUserPool](#) API operation. The same approach, with different input fields, applies to [UpdateUserPoolClient](#).

Important

If you don't provide values for existing parameters, Amazon Cognito sets them to default values. For example, when you have existing LambdaConfig and you submit an `UpdateUserPool` with an empty `LambdaConfig`, you delete the assignment of all Lambda functions to user pool triggers. Plan accordingly when you want to automate changes to your user pool configuration.

1. Capture the existing state of your user pool with [DescribeUserPool](#).
2. Format the output of `DescribeUserPool` to match the [request parameters](#) of `UpdateUserPool`. Remove the following top-level fields and their child objects from the output JSON.
 - `Arn`
 - `CreationDate`
 - `CustomDomain`
 - Update this field with the [UpdateUserPoolDomain](#) API operation.
 - `Domain`
 - Update this field with the [UpdateUserPoolDomain](#) API operation.
 - `EmailConfigurationFailure`
 - `EstimatedNumberOfUsers`
 - `Id`
 - `LastModifiedDate`
 - `Name`
 - `SchemaAttributes`
 - `SmsConfigurationFailure`
 - `Status`
3. Confirm that the resulting JSON matches the [request parameters](#) of `UpdateUserPool`.
4. Modify any parameters that you want to change in the resulting JSON.
5. Submit an `UpdateUserPool` API request with your modified JSON as the request input.

You can also use this modified `DescribeUserPool` output in the `--cli-input-json` parameter of `update-user-pool` in the AWS CLI.

Alternately, run the following AWS CLI command to generate JSON with blank values for the accepted input fields for `update-user-pool`. You can then populate these fields with the existing values from your user pool.

```
aws cognito-idp update-user-pool --generate-cli-skeleton --output json
```

Run the following command to generate the same JSON object for an app client.

```
aws cognito-idp update-user-pool-client --generate-cli-skeleton --output json
```

Application-specific settings with app clients

A user pool app client is a configuration within a user pool that interacts with one mobile or web application that authenticates with Amazon Cognito. App clients can call authenticated and unauthenticated API operations, and read or modify some or all of your users' attributes. Your app must identify itself to the app client in operations to register, sign in, and handle forgotten passwords. These API requests must include self-identification with an app client ID, and authorization with an optional client secret. You must secure any app client IDs or secrets so that only authorized client apps can call these unauthenticated operations. Additionally, if you configure your app to sign authenticated API requests with AWS credentials, you must secure your credentials against user inspection.

You can create multiple apps for a user pool. An app client might be linked to the code platform of an app, or a separate tenant in your user pool. 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.

You can apply settings for the following user pool features at the app client level:

1. [Analytics](#)
2. [Managed login](#) IdPs, grant types, callback URLs, and customization
3. [Resource servers and custom scopes](#)
4. [Threat protection](#)
5. [Attribute read and write permissions](#)
6. [Token expiration and revocation](#)

7. [Authentication flows](#)

App client types

When you create an app client in Amazon Cognito, you can pre-populate options based on the standard OAuth client types **public client** and **confidential client**. Configure a **confidential client** with a **client secret**. For more information about client types, see [IETF RFC 6749 #2.1](#).

Public client

A public client runs in a browser or on a mobile device. Because it does not have trusted server-side resources, it does not have a client secret.

Confidential client

A confidential client has server-side resources that can be trusted with a **client secret** for unauthenticated API operations. The app might run as a daemon or shell script on your backend server.

Client secret

A client secret, or client password, is a fixed string that your app must use in all API requests to the app client. Your app client must have a client secret to perform `client_credentials` grants. For more information, see [IETF RFC 6749 #2.3.1](#).

You can't change secrets after you create an app. You can create a new app with a new secret if you want to rotate the secret. You can also delete an app to block access from apps that use that app client ID.

You can use a confidential client, and a client secret, with a public app. Use an Amazon CloudFront proxy to add a SECRET_HASH in transit. For more information, see [Protect public clients for Amazon Cognito by using an Amazon CloudFront proxy](#) on the AWS blog.

JSON web tokens

Amazon Cognito app clients can issue JSON web tokens (JWTs) of the following types.

Identity (ID) token

A verifiable statement that your user is authenticated from your user pool. OpenID Connect (OIDC) added the [ID token specification](#) to the access and refresh token standards defined by

OAuth 2.0. The ID token contains identity information, like user attributes, that your app can use to create a user profile and provision resources. See [Understanding the identity \(ID\) token](#) for more information.

Access token

A verifiable statement of your user's access rights. The access token contains [scopes](#), a feature of OIDC and OAuth 2.0. Your app can present scopes to back-end resources and prove that your user pool authorized a user or machine to access data from an API, or their own user data. An access token with *custom scopes*, often from an M2M client-credentials grant, authorizes access to a resource server. See [Understanding the access token](#) for more information.

Refresh token

An encrypted statement of initial authentication that your app can present to your user pool when your user's tokens expire. A refresh-token request returns new, unexpired access and ID tokens. See [Understanding the refresh token](#) for more information.

You can set the expiration of these tokens for each app client from the **App clients** menu of your user pool in the [Amazon Cognito console](#).

App client terms

The following terms are available properties of app clients in the Amazon Cognito console.

Allowed callback URLs

A callback URL indicates where the user will be redirected after a successful sign-in. Choose at least one callback URL. The callback URL must:

- 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.

Allowed sign out URLs

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

Attribute read and write permissions

Your user pool might have many customers, each with their own app client and IdPs. You can configure your app client to have read and write access to only those user attributes that are relevant to the app. In cases like machine-to-machine (M2M) authorization, you can grant access to none of your user attributes.

Considerations for attribute read and write permissions configuration

- When you create an app client and don't customize attribute read and write permissions, Amazon Cognito grants read and write permissions to all user pool attributes.
- You can grant write access to immutable [custom attributes](#). Your app client can write values to immutable attributes when you create or sign up a user. After this, you can't write values to any immutable custom attributes for the user.
- App clients must have write access to required attributes in your user pool. The Amazon Cognito console automatically sets required attributes as writeable.
- You can't permit an app client to have write access to `email_verified` or `phone_number_verified`. A user pool administrator can modify these values. A user can only change the value of these attributes through [attribute verification](#).

Authentication flows

The methods that your app client allows for sign-in. Your app can support authentication with username and password, Secure Remote Password (SRP), custom authentication with Lambda triggers, and token refresh. As a best security practice, use SRP authentication as your primary sign-in method. Managed login automatically signs users in with SRP.

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/scoped`. See [Scopes, M2M, and APIs with resource servers](#).

Default redirect URI

Replaces the `redirect_uri` parameter in authentication requests for users with third-party IdPs. Configure this app client setting with the `DefaultRedirectURI` parameter of a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) API request. This URL must also be a member of

the CallbackURLs for your app client. Amazon Cognito redirects authenticated sessions to this URL when:

1. Your app client has one [identity provider](#) assigned and multiple [callback URLs](#) defined. Your user pool redirects authentication requests to the [authorization server](#) to the default redirect URI when they don't include a `redirect_uri` parameter.
2. Your app client has one [identity provider](#) assigned and one [callback URLs](#) defined. In this scenario it's not necessary to define a default callback URL. Requests that don't include a `redirect_uri` parameter redirect to the one available callback URL.

Identity providers

You can choose some or all of your user pool external identity providers (IdPs) to authenticate your users. Your app client can also authenticate only local users in your user pool. When you add an IdP to your app client, you can generate authorization links to the IdP and display it on your managed login sign-in page. You can assign multiple IdPs, but you must assign at least one. For more information on using external IdPs, see [User pool sign-in with a third party identity provider](#).

OpenID Connect scopes

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

- The `openid` scope declares that you want to retrieve an ID token and a user's unique ID. It also requests all or some user attributes, depending on additional scopes in the request. Amazon Cognito doesn't return an ID token unless you request the `openid` scope. The `openid` scope authorizes structural ID token claims like expiration and key ID, and determines the user attributes that you receive in a response from the [userInfo endpoint](#).
 - When `openid` is the only scope that you request, Amazon Cognito populates the ID token with all user attributes that the current app client can read. The `userInfo` response to an access token with this scope alone returns all user attributes.
 - When you request `openid` with other scopes like `phone`, `email`, or `profile`, the ID token and `userInfo` return the user's unique ID and the attributes defined by the additional scopes.
- 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 `aws.cognito.signin.user.admin` scope grants access to [Amazon Cognito user pools 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.

For more information about scopes, see the list of [standard OIDC scopes](#).

OAuth grant types

An OAuth grant is a method of authentication that retrieves user-pool tokens. Amazon Cognito supports the following types of grants. To integrate these OAuth grants in your app, you must add a domain to your user pool.

Authorization code grant

The authorization code grant generates a code that your app can exchange for user pool tokens with the [Token endpoint](#). When you exchange an authorization code, your app receives ID, access, and refresh tokens. This OAuth flow, like the implicit grant, happens in your users' browsers. An authorization code grant is the most secure grant that Amazon Cognito offers, because tokens aren't visible in your users' sessions. Instead, your app generates the request that returns tokens and can cache them in protected storage. For more information, see [Authorization code in IETF RFC 6749 #1.3.1](#)

Note

As a best security practice in public-client apps, activate only the authorization-code grant OAuth flow, and implement Proof Key for Code Exchange (PKCE) to restrict token exchange. With PKCE, a client can only exchange an authorization code when they have provided the token endpoint with the same secret that was presented in the original authentication request. For more information on PKCE, see [IETF RFC 7636](#).

Implicit grant

The implicit grant delivers an access and ID token, but not refresh token, to your user's browser session directly from the [Authorize endpoint](#). An implicit grant removes the requirement for a separate request to the token endpoint, but isn't compatible with PKCE and doesn't return refresh tokens. This grant accommodates testing scenarios and app architecture that can't

complete authorization-code grants. For more information, see [Implicit grant in IETF RFC 6749 #1.3.2](#). You can activate both the authorization-code grant and the implicit grant in an app client, and then use each grant as needed.

Client credentials grant

The client credentials grant is for machine-to-machine (M2M) communications. Authorization-code and implicit grants issue tokens to authenticated human users. Client credentials grant scope-based authorization from a non-interactive system to an API. Your app can request client credentials directly from the token endpoint and receive an access token. For more information, see [Client Credentials in IETF RFC 6749 #1.3.4](#). You can only activate client-credentials grants in app clients that have a client secret and that don't support authorization-code or implicit grants.

Note

Because you don't invoke the client credentials flow as a user, this grant can only add *custom* scopes to access tokens. A custom scope is one that you define for your own resource server. Default scopes like `openid` and `profile` don't apply to nonhuman users.

Because ID tokens are a validation of user attributes, they aren't relevant to M2M communication, and a client credentials grants doesn't issue them. See [Scopes, M2M, and APIs with resource servers](#).

Client credentials grants add costs to your AWS bill. For more information, see [Amazon Cognito Pricing](#).

Creating an app client

AWS Management Console

To create an app client (console)

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool. Both options prompt you to configure an app client with application-specific settings.

4. Choose an **Application type** that reflects your application architecture.
5. **Name your application** with a friendly identifier.
6. Enter a **Return URL**.
7. Choose **Create app client**. Or, to set detailed options, **Switch to classic settings**. You can also change advanced options after you create your app client.
8. Amazon Cognito returns you to app client details. To access example code for your application, select a platform from the **Quick setup guide** tab.

AWS CLI

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

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]"
```

See the AWS CLI command reference for more information: [create-user-pool-client](#)

Amazon Cognito user pools API

Generate a [CreateUserPoolClient](#) API request. You must specify a value for all parameters that you don't want set to a default value.

Updating a user pool app client (AWS CLI and AWS API)

At the AWS CLI, enter the following command:

```
aws cognito-identity 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,  
        "AuthSessionValidity": 3,  
        "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](#)

Getting information about a user pool app client (AWS CLI and AWS API)

```
aws cognito-identity 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](#)

Working with user devices in your user pool

When you sign in local user pool users with the Amazon Cognito user pools API, you can associate your users' activity logs from [threat protection](#) with each of their devices and, optionally, allow your users to skip multi-factor authentication (MFA) if they're on a trusted device. Amazon Cognito includes a device key in the response to any sign-in that doesn't already include device information. The device key is in the format *Region_UUID*. With a device key, a Secure Remote Password (SRP) library, and a user pool that permits device authentication, you can prompt users in your app to trust the current device and no longer prompt for an MFA code at sign-in.

Topics

- [Setting up remembered devices](#)
- [Getting a device key](#)
- [Signing in with a device](#)
- [Viewing, updating and forgetting devices](#)

Setting up remembered devices

With Amazon Cognito user pools, you can associate each of your users' devices with a unique device identifier: a device key. When you present the device key and perform device authentication at sign-in, you can take advantage of two features.

- With advanced security features, you can monitor user activity on specific devices for security and analytics purposes. When users sign in, your app has the option to authenticate each user and their device, adding device information to their activity logs.
- Remembering devices also supports a *trusted device* authentication flow, where your users can choose to sign in without MFA until a later time, as determined by the security requirements of your app or the preferences of your users. For example, your application might prompt your users to trust a device for 30, 60, or 90 days. You can store this date in a custom attribute and on that date, change the remembered status of their device. You must then re-prompt your user to submit an MFA code and set the device to be remembered again after successful authentication.

Remembered devices can override MFA only in user pools with MFA active.

When your user signs in with a remembered device, you must perform an additional device authentication during their authentication flow. For more information, see [Signing in with a device](#).

Configure your user pool to remember devices in the **Sign-in** menu of your user pool, under **Device tracking**. When setting up the remembered devices functionality through the Amazon Cognito console, you have three options: **Always**, **User Opt-In**, and **No**.

Don't remember

Your user pool doesn't prompt users to remember devices when they sign in.

Always remember

When your app confirms a user's device, your user pool always remembers the device and doesn't return MFA challenges on future successful device sign-ins.

User opt-in

When your app confirms a user's device, your user pool doesn't automatically suppress MFA challenges. You must prompt your user to choose whether they want to remember the device.

When you choose **Always remember** or **User Opt-In**, Amazon Cognito generates a device-identifier key and secret every time a user signs in from an unidentified device. The device key is the initial identifier that your app sends to your user pool when your user performs device authentication.

With each confirmed user device, whether remembered automatically or opted-in, you can use the device-identifier key and secret to authenticate a device on every user sign-in.

You can also configure remembered-device settings for your user pool in a [CreateUserPool](#) or [UpdateUserPool](#) API request. For more information, see the [DeviceConfiguration](#) property.

The Amazon Cognito user pools API has additional operations for remembered devices.

1. [ListDevices](#) and [AdminListDevices](#) return a list of the device keys and their metadata for a user.
2. [GetDevice](#) and [AdminGetDevice](#) return the device key and metadata for a single device.
3. [UpdateDeviceStatus](#) and [AdminUpdateDeviceStatus](#) set a user's device as remembered or not remembered.
4. [ForgetDevice](#) and [AdminForgetDevice](#) remove a user's confirmed device from their profile.

API operations with names that begin with Admin are for use in server-side apps and must be authorized with IAM credentials. For more information, see [Understanding API, OIDC, and managed login pages authentication](#).

Getting a device key

Any time that your user signs in with the user pools API and doesn't include a device key in the authentication parameters as DEVICE_KEY, Amazon Cognito returns a new device key in the response. In your public client-side app, place the device key in app storage so that you can include it in future requests. In your confidential server-side app, set a browser cookie or another client-side token with your user's device key.

Before your user can sign in with their trusted device, your app must confirm the device key and provide additional information. Generate a [ConfirmDevice](#) request to Amazon Cognito that confirms your user's device with the device key, a friendly name, password verifier, and a salt. If you configured your user pool for opt-in device authentication, Amazon Cognito responds to your ConfirmDevice request with a prompt that your user must choose whether to remember the current device. Respond with your user's selection in an [UpdateDeviceStatus](#) request.

When you confirm your user's device but don't set it as remembered, Amazon Cognito stores the association but proceeds with non-device sign-in when you provide the device key. Devices can generate logs that are useful for user security and troubleshooting. A confirmed but unremembered device doesn't take advantage of the sign-in feature, but does take advantage of the security monitoring logs feature. When you activate advanced security features for your app client and encode a device footprint into your request, Amazon Cognito associates user events with the confirmed device.

To get a new device key

1. Start your user's sign-in session with an [InitiateAuth](#) API request.
2. Respond to all authentication challenges with [RespondToAuthChallenge](#) until you receive JSON web tokens (JWTs) that mark your user's sign-in session complete.
3. In your app, record the values that Amazon Cognito returns in NewDeviceMetadata in its RespondToAuthChallenge or InitiateAuth response: DeviceGroupKey and DeviceKey.
4. Generate a new SRP secret for your user: a salt and a password verifier. This function is available in SDKs that provide SRP libraries.
5. Prompt your user for a device name, or generate one from your user's device characteristics.
6. Provide your user's access token, device key, device name, and SRP secret in a [ConfirmDevice](#) API request. If your user pool is set to **Always remember** devices, your user's registration is complete.
7. If Amazon Cognito responded to ConfirmDevice with "UserConfirmationNecessary": true, prompt your user to choose if they would like to remember the device. If they affirm that they want to remember the device, generate an [UpdateDeviceStatus](#) API request with your user's access token, device key, and "DeviceRememberedStatus": "remembered".
8. If you have instructed Amazon Cognito to remember the device, the next time they sign in, instead of an MFA challenge, they're presented with a DEVICE_SRP_AUTH challenge.

Signing in with a device

After you configure a user's device to be remembered, Amazon Cognito no longer requires them to submit an MFA code when they sign in with the same device key. Device authentication only replaces the MFA-authentication challenge with a device-authentication challenge. You can't sign users in with device authentication only. Your user must first complete authentication with their password or a custom challenge. The following is the authentication process for a user on a remembered device.

To perform device authentication in a flow that uses [Custom authentication challenge Lambda triggers](#), pass a DEVICE_KEY parameter in your [InitiateAuth](#) API request. After your user succeeds all challenges and the CUSTOM_CHALLENGE challenge returns an issueTokens value of true, Amazon Cognito returns one final DEVICE_SRP_AUTH challenge.

To sign in with a device

1. Retrieve your user's device key from client storage.

2. Start your user's sign-in session with an [InitiateAuth](#) API request. Choose an AuthFlow of USER_SRP_AUTH, REFRESH_TOKEN_AUTH, USER_PASSWORD_AUTH, or CUSTOM_AUTH. In AuthParameters, add your user's device key to the DEVICE_KEY parameter, and include the other required parameters for your selected sign-in flow.
 - a. You can also pass DEVICE_KEY in the parameters of a PASSWORD_VERIFIER response to an authentication challenge.
3. Complete challenge responses until you receive a DEVICE_SRP_AUTH challenge in the response.
4. In a [RespondToAuthChallenge](#) API request, send a ChallengeName of DEVICE_SRP_AUTH and parameters for USERNAME, DEVICE_KEY, and SRP_A.
5. Amazon Cognito responds with a DEVICE_PASSWORD_VERIFIER challenge. This challenge response includes values for SECRET_BLOCK and SRP_B.
6. With your SRP library, generate and submit PASSWORD_CLAIM_SIGNATURE, PASSWORD_CLAIM_SECRET_BLOCK, TIMESTAMP, USERNAME, and DEVICE_KEY parameters. Submit these in an additional RespondToAuthChallenge request.
7. Complete additional challenges until you receive the user's JWTs.

The following pseudocode demonstrates how to calculate values for your DEVICE_PASSWORD_VERIFIER challenge response.

```
PASSWORD_CLAIM_SECRET_BLOCK = SECRET_BLOCK
TIMESTAMP = Tue Sep 25 00:09:40 UTC 2018
PASSWORD_CLAIM_SIGNATURE = Base64(SHA256_HMAC(K_USER, DeviceGroupKey + DeviceKey +
    PASSWORD_CLAIM_SECRET_BLOCK + TIMESTAMP))
K_USER = SHA256_HASH(S_USER)
S_USER = (SRP_B - k * gx)(a + ux)
x = SHA256_HASH(salt + FULL_PASSWORD)
u = SHA256_HASH(SRP_A + SRP_B)
k = SHA256_HASH(N + g)
```

Viewing, updating and forgetting devices

You can implement the following features in your app with the Amazon Cognito API.

1. Display information about a user's current device.
2. Display a list of all of your user's devices.
3. Forget a device.

4. Update a device remembered state.

The access tokens that authorize the API requests in the following descriptions must include the `aws.cognito.signin.user.admin` scope. Amazon Cognito adds a claim for this scope to all access tokens that you generate with the Amazon Cognito user pools API. Third-party IdPs must separately manage devices and MFA for their users who authenticate to Amazon Cognito. In managed login, you can request the `aws.cognito.signin.user.admin` scope, but managed login automatically adds device information to advanced security user logs, and doesn't offer to remember devices.

Display information about a device

You can query information about a user's device to determine if it is still in current use. For example, you might want to deactivate remembered devices after they haven't signed in for 90 days.

- To display your user's device information in a public-client app, submit your user's access key and device key in a [GetDevice](#) API request.
- To display your user's device information in a confidential-client app, sign an [AdminGetDevice](#) API request with AWS credentials and submit your user's username, device key, and user pool.

Display a list of all your user's devices

You can display a list of all your user's devices and their properties. For example, you might want to verify that the current device matches a remembered device.

- In a public-client app, submit your user's access token in a [ListDevices](#) API request.
- In a confidential-client app, sign an [AdminListDevices](#) API request with AWS credentials and submit your user's username and user pool.

Forget a device

You can delete a user's device key. You might want to do this when you determine that your user no longer uses a device, or when you detect unusual activity and want to prompt a user to complete MFA again. To register the device again later, you must generate and store a new device key.

- In a public-client app, submit your user's device key and access token in [ForgetDevice](#) API request.
- In a confidential-client app, submit your user's device key and access token in [AdminForgetDevice](#) API request.

Scopes, M2M, and APIs with resource servers

After you configure a domain for your user pool, Amazon Cognito automatically provisions an OAuth 2.0 authorization server and a hosted web UI with sign-up and sign-in pages that your app can present to your users. For more information see [User pool managed login](#). You can choose the scopes that you want the authorization server to add to access tokens. Scopes authorize access to resource servers and user data.

A *resource server* is an OAuth 2.0 API server. To secure access-protected resources, it validates that access tokens from your user pool contain the scopes that authorize the requested method and path in the API that it protects. It verifies the issuer based on the token signature, validity based on token expiration time, and access level based on the scopes in token claims. User pool scopes are in the access token scope claim. For more information about the claims in Amazon Cognito access tokens, see [Understanding the access token](#).

With Amazon Cognito, the scopes in access tokens can authorize access to external APIs or to user attributes. You can issue access tokens to local users, federated users, or machine identities.

Topics

- [API authorization](#)
- [Machine-to-machine \(M2M\) authorization](#)
- [About scopes](#)
- [About resource servers](#)

API authorization

The following are some of the ways that you can authorize requests to APIs with Amazon Cognito tokens:

Access token

When you add an Amazon Cognito authorizer to a REST API method request configuration, add **Authorization scopes** to the authorizer configuration. With this configuration, your API accepts access tokens in the Authorization header and reviews them for accepted scopes.

ID token

When you pass a valid ID token to an Amazon Cognito authorizer in your REST API, API Gateway accepts the request and passes the ID token contents to the API backend.

Amazon Verified Permissions

In Verified Permissions, you have the option to create an [API-linked policy store](#). Verified Permissions creates and assigns a Lambda authorizer that processes ID or access tokens from your request Authorization header. This Lambda authorizer passes your token to your policy store, where Verified Permissions compares it to policies and returns an allow or deny decision to the authorizer.

More resources

- [Controlling and managing access to a REST API in API Gateway](#)
- [Authorization with Amazon Verified Permissions](#)

Machine-to-machine (M2M) authorization

Amazon Cognito supports applications that access API data with *machine identities*. Machine identities in user pools are [confidential clients](#) that run on application servers and connect to remote APIs. Their operation happens without user interaction: scheduled tasks, data streams, or asset updates. When these clients authorize their requests with an access token, they perform *machine to machine*, or M2M, authorization. In M2M authorization, a shared secret replaces user credentials in access control.

An application that accesses an API with M2M authorization must have a client ID and client secret. In your user pool, you must build an app client that supports client credentials grants. To support client credentials, your app client must have a client secret and you must have a user pool domain. In this flow, your machine identity requests an access token directly from the [Token endpoint](#). You can authorize only custom scopes from [resource servers](#) in access tokens for client credentials grants. For more information about setting up app clients, see [Application-specific settings with app clients](#).

The access token from a client credentials grant is a verifiable statement of the operations that you want to permit your machine identity to request from an API. To learn more about how access tokens authorize API requests, continue reading. For an example application, see [Amazon Cognito and API Gateway based machine to machine authorization using AWS CDK](#).

M2M authorization has a billing model that differs from the way that monthly active users (MAUs) are billed. Where user authentication carries a cost per active user, M2M billing reflects active client credentials app clients and total token-request volume. For more information, see [Amazon Cognito Pricing](#). To control costs for M2M authorization, optimize the duration of access tokens and the number of token requests that your applications make. See [Managing user pool token expiration and caching](#) for a way to use API Gateway caching to reduce requests for new tokens in M2M authorization.

For information about optimizing Amazon Cognito operations that add costs to your AWS bill, see [Managing costs](#).

About scopes

A *scope* is a level of access that an app can request to a resource. In an Amazon Cognito access token, the scope is backed up by the trust that you set up with your user pool: a trusted issuer of access tokens with a known digital signature. User pools can generate access tokens with scopes that prove your customer is allowed to manage some or all of their own user profile, or to retrieve data from a back-end API. Amazon Cognito user pools issue access tokens with *the user pools reserved API scope, custom scopes, and OpenID Connect (OIDC) scopes*.

The user pools reserved API scope

The `aws.cognito.signin.user.admin` scope authorizes self-service operations for the current user in the Amazon Cognito user pools API. It authorizes the bearer of an access token to query and update all information about the bearer with, for example, the [GetUser](#) and [UpdateUserAttributes](#) API operations. When you authenticate your user with the Amazon Cognito user pools API, this is the only scope you receive in the access token. It's also the only scope you need to read and write user attributes that you've authorized your app client to read and write. You can also request this scope in requests to your [Authorize endpoint](#). This scope alone isn't sufficient to request user attributes from the [userInfo endpoint](#). For access tokens that authorize both user pools API and `userInfo` requests for your users, you must request both of the scopes `openid` and `aws.cognito.signin.user.admin` in an `/oauth2/authorize` request.

Custom scopes

Custom scopes authorize requests to the external APIs that resource servers protect. You can request custom scopes with other types of scopes. You can find more information about custom scopes throughout this page.

OpenID Connect (OIDC) scopes

When you authenticate users with your user pool authorization server, including with managed login, you must request scopes. You can authenticate user pool local users and third-party federated users in your Amazon Cognito authorization server. OIDC scopes authorize your app to read user information from the [userInfo endpoint](#) of your user pool. The OAuth model, where you query user attributes from the userInfo endpoint, can optimize your app for a high volume of requests for user attributes. The userInfo endpoint returns attributes at a permission level that's determined by the scopes in the access token. You can authorize your app client to issue access tokens with the following OIDC scopes.

openid

The minimum scope for OpenID Connect (OIDC) queries. Authorizes the ID token, the unique-identifier claim sub, and the ability to request other scopes.

Note

When you request the openid scope and no others, your user pool ID token and userInfo response include claims for all user attributes that your app client can read. When you request openid and other OIDC scopes like profile, email, and phone, the contents of the ID token and [userInfo](#) response are limited to the constraints of the additional scopes.

For example, a request to the [Authorize endpoint](#) with the parameter `scope=openid +email` returns an ID token with sub, email, and email_verified. The access token from this request returns the same attributes from [userInfo endpoint](#). A request with parameter `scope=openid` returns all client-readable attributes in the ID token and from userInfo.

profile

Authorizes all user attributes that the app client can read.

email

Authorizes the user attributes `email` and `email_verified`. Amazon Cognito returns `email_verified` if it has had a value explicitly set.

phone

Authorizes the user attributes `phone_number` and `phone_number_verified`.

About resource servers

A resource server API might grant access to the information in a database, or control your IT resources. An Amazon Cognito access token can authorize access to APIs that support OAuth 2.0. Amazon API Gateway REST APIs have [built-in support](#) for authorization with Amazon Cognito access tokens. Your app passes the access token in the API call to the resource server. The resource server inspects the access token to determine if access should be granted.

Amazon Cognito might make future updates to the schema of user pool access tokens. If your app analyzes the contents of the access token before it passes it to an API, you must engineer your code to accept updates to the schema.

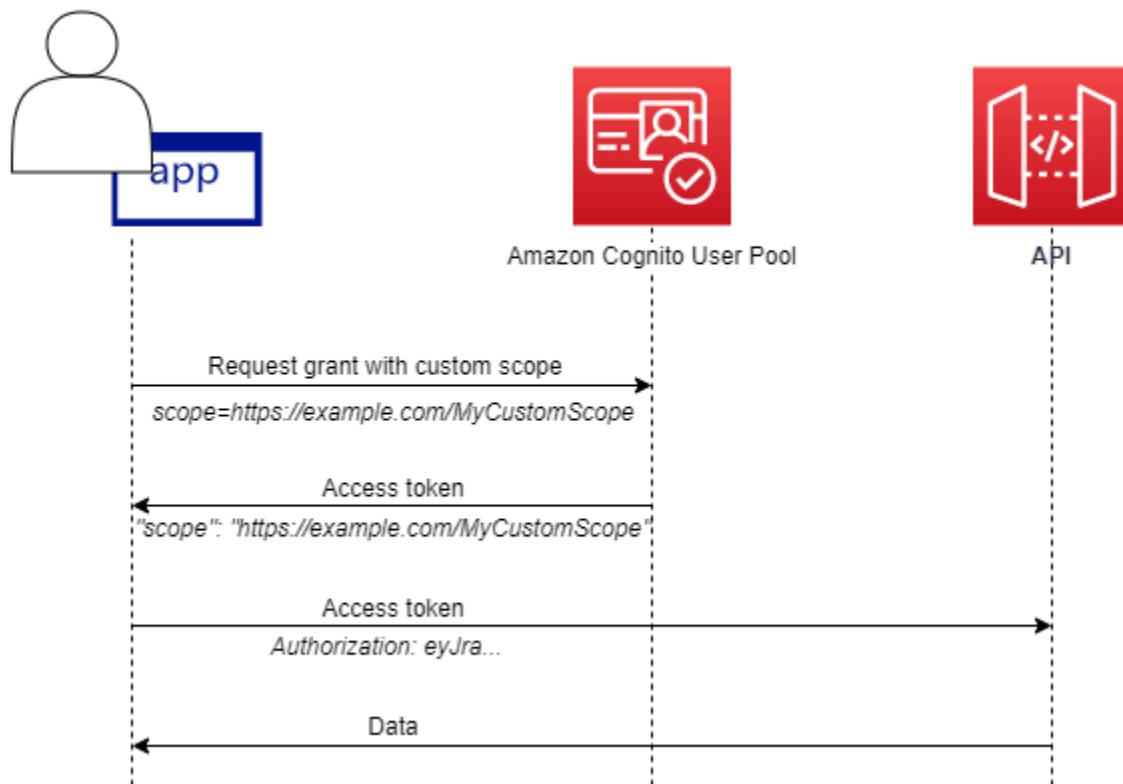
Custom scopes are defined by you, and extend the authorization capabilities of a user pool to include purposes unrelated to querying and modifying users and their attributes. For example, if you have a resource server for photos, it might define two scopes: `photos.read` for read access to the photos and `photos.write` for write/delete access. You can configure an API to accept access tokens for authorization, and grant HTTP GET requests to access tokens with `photos.read` in the scope claim, and HTTP POST requests to tokens with `photos.write`. These are *custom scopes*.

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 tokens, see [Verifying a JSON Web Token](#). For more information about verifying and using user pool tokens in Amazon API Gateway, see the blog [Integrating Amazon Cognito User Pools with API Gateway](#). 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](#).

Overview

With Amazon Cognito, you can create OAuth 2.0 **Resource servers** and associate **Custom scopes** with them. Custom scopes in an access token authorize specific actions in your API. You can authorize any app client in your user pool to issue custom scopes from any of your resource servers. Associate your custom scopes with an app client and request those scopes in OAuth 2.0 authorization code grants, implicit grants, and client credentials grants from the [Token endpoint](#). Amazon Cognito adds custom scopes to the scope claim in an access token. A client can use the access token against its resource server, which makes the authorization decision based on the scopes present in the token. For more information about access token scope, see [Using Tokens with User Pools](#).



To get an access token with custom scopes, your app must make a request to the [Token endpoint](#) to redeem an authorization code or to request a client credentials grant. In managed login, you can also request custom scopes in an access token from an implicit grant.

Note

Because they are designed for human-interactive authentication with the user pool as the IdP, [InitiateAuth](#) and [AdminInitiateAuth](#) requests only produce a scope claim in the access token with the single value `aws.cognito.signin.user.admin`.

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.

- **Resource server name:** A friendly name for the resource server, such as Solar system object tracker or Photo API.
- **Resource server identifier:** A unique identifier for the resource server. The identifier is any name that you want to associate with your API, for example `solar-system-data`. You can configure longer identifiers like `https://solar-system-data-api.example.com` as a more direct reference to API URI paths, but longer strings increase the size of access tokens.
- **Scope name:** The value that you want in your scope claims. For example, `sunproximity.read`.
- **Description:** A friendly description of the scope. For example, Check current proximity to sun.

Amazon Cognito can include custom scopes in access tokens for any users, whether they are local to your user pool or federated with a third-party identity provider. You can choose scopes for your users' access tokens during authentication flows with the OAuth 2.0 authorization server that includes managed login. Your user's authentication must begin at the [Authorize endpoint](#) with scope as one of the request parameters. The following is a recommended format for resource servers. For an identifier, use an API friendly name. For a custom scope, use the action that they authorize.

resourceServerIdentifier/scopedName

For example, you've discovered a new asteroid in the Kuiper belt and you want to register it through your `solar-system-data` API. The scope that authorizes write operations to the database of asteroids is `asteroids.add`. When you request the access token that will authorize

you to register your discovery, format your scope HTTPS request parameter as `scope=solar-system-data/asteroids.add`.

Deleting a scope from a resource server does not delete its association with all clients. Instead, the scope is marked *inactive*. Amazon Cognito doesn't add inactive scopes to access tokens, but otherwise proceeds as normal if your app requests one. If you add the scope to your resource server again later, then Amazon Cognito again writes it to the access token. If you request a scope that you haven't associated with your app client, regardless of whether you deleted it from your user pool resource server, authentication fails.

You can use the AWS Management Console, API, or CLI to define resource servers and scopes for your user pool.

Defining a resource server for your user pool (AWS Management Console)

You can use the AWS Management Console to define a resource server for your user pool.

To define a resource server

1. Sign in to the [Amazon Cognito console](#).
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Domain** menu under **Branding** and locate **Resource servers**.
4. Choose **Create a resource server**.
5. Enter a **Resource server name**. For example, Photo Server.
6. Enter a **Resource server identifier**. For example, com.example.photos.
7. Enter **Custom scopes** for your resources, such as `read` and `write`.
8. For each **Scope name**, enter a **Description**, such as view your photos and update your photos.
9. Choose **Create**.

Your custom scopes can be reviewed in the **Domain** menu under **Resource servers**, in the **Custom scopes** column. Custom scopes can be enabled for app clients from the **App clients** menu under **Applications**. Select an app client, locate **Login pages** and choose **Edit**. Add **Custom scopes** and choose **Save changes**.

Defining a resource server for your user pool (AWS CLI and AWS API)

Use the following commands to specify resource server settings for your user pool.

To create a resource server

- AWS CLI: aws cognito-idp create-resource-server
- AWS API: [CreateResourceServer](#)

To get information about your resource server settings

- AWS CLI: aws cognito-idp describe-resource-server
- AWS API: [DescribeResourceServer](#)

To list information about all resource servers for your user pool

- AWS CLI: aws cognito-idp list-resource-servers
- AWS API: [ListResourceServers](#)

To delete a resource server

- AWS CLI: aws cognito-idp delete-resource-server
- AWS API: [DeleteResourceServer](#)

To update the settings for a resource server

- AWS CLI: aws cognito-idp update-resource-server
- AWS API: [UpdateResourceServer](#)

Using Amazon Pinpoint for user pool analytics

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 custom attributes for your app. 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** configuration for your app client in the **App clients** menu in the Amazon Cognito console, Amazon Pinpoint creates additional endpoints for user email addresses and phone numbers.

When you activate Amazon Pinpoint analytics in your user pool with the Amazon Cognito console, you also create a [service-linked role](#) that Amazon Cognito assumes when it makes an API request to Amazon Pinpoint for your user pool. The IAM principal that adds your analytics configuration must have [CreateServiceLinkedRole](#) permissions. The service-linked role is [AWSServiceRoleForAmazonCognitoldp](#). For more information, see [Using service-linked roles for Amazon Cognito](#).

When you apply an `AnalyticsConfiguration` to your app client in the Amazon Cognito API, you can assign a custom IAM role for Amazon Pinpoint and an external ID to assume the role. The role must trust the `cognito-idp` service principal, and if the role trust policy requires an external ID, it must match your `AnalyticsConfiguration`. You must grant the role `cognito-idp:Describe*` permissions, and the following permissions for your **Amazon Pinpoint project**.

- `mobiletargeting:UpdateEndpoint`
- `mobiletargeting:PutEvents`

Amazon Cognito and Amazon Pinpoint Region availability

The following table shows the AWS Region mappings between Amazon Cognito and Amazon Pinpoint that meet one of the following conditions.

- You can only use an Amazon Pinpoint project in the US East (N. Virginia) (`us-east-1`) Region.
- You can use an Amazon Pinpoint project in the same Region or in the US East (N. Virginia) (`us-east-1`) Region

By default, Amazon Cognito can only send analytics to a Amazon Pinpoint project in the same AWS Region. The exceptions to this rule are the Regions in the following table, and Regions where Amazon Pinpoint is unavailable.

Amazon Pinpoint isn't available in the following Regions. Amazon Cognito user pools in these Regions don't support analytics.

- Europe (Milan)
- Middle East (Bahrain)
- Asia Pacific (Osaka)
- Israel (Tel Aviv)
- Africa (Cape Town)
- Asia Pacific (Jakarta)

The table shows the relation between the Region where you built your Amazon Cognito user pool and the corresponding Region in Amazon Pinpoint. You must configure your Amazon Pinpoint project in an available Region to integrate it with Amazon Cognito.

Amazon Cognito user pool Region	Region for Amazon Pinpoint project
ap-northeast-1	us-east-1
ap-northeast-2	us-east-1
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-northeast-1, you can create your Amazon Pinpoint project in us-east-1.
- If you create a user pool in ap-south-1, you can create your Amazon Pinpoint project in either us-east-1 or ap-south-1.

Note

For all AWS Regions except those in the preceding table, Amazon Cognito can only use an Amazon Pinpoint project in the same Region as your user pool. If Amazon Pinpoint isn't available in the Region where you built your user pool, and it's not listed in the table, then Amazon Cognito doesn't support Amazon Pinpoint analytics in that Region. For detailed AWS Region information, see [Amazon Pinpoint endpoints and quotas](#).

Specifying Amazon Pinpoint analytics settings (AWS Management Console)

You can configure your Amazon Cognito user pool to send analytics data to Amazon Pinpoint. Amazon Cognito only sends analytics data to Amazon Pinpoint for local users. After you configure your user pool to associate with a Amazon Pinpoint project, you must include `AnalyticsMetadata` in your API requests. For more information, see [Integrating your app with Amazon Pinpoint](#).

To specify analytics settings

1. Go to the [Amazon Cognito console](#). You might be prompted for your AWS credentials.
2. Select **User Pools** and choose an existing user pool from the list.
3. Choose the **App clients** menu and select the app client that you want to update.
4. In the **Analytics** tab under **Pinpoint analytics**, choose **Enable**.
5. Choose a **Pinpoint Region**.
6. Choose an **Amazon Pinpoint project** or select **Create Amazon Pinpoint project**.

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. Under **User data sharing**, choose **Share user data with Amazon Pinpoint** if you want Amazon Cognito to send email addresses and phone numbers to Amazon Pinpoint and create additional endpoints for users. After your users verify their email address and phone number, Amazon Cognito only shares them 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**.

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`

Integrating your app with Amazon Pinpoint

You can publish analytics metadata to Amazon Pinpoint for Amazon Cognito *local users* in the *user pools API*.

Local users

Users who signed up for an account or were created in your user pool instead of signing in through a third-party identity provider (IdP).

User pools API

The operations that you can integrate with an AWS SDK, using an app with a custom user interface (UI). You can't pass analytics metadata for federated or local users who sign in through managed login. See the [Amazon Cognito API Reference](#) for a list of user pools API operations.

After you configure your user pool to publish to a campaign, Amazon Cognito passes metadata to Amazon Pinpoint for the following API operations.

- `AdminInitiateAuth`
- `AdminRespondToAuthChallenge`
- `ConfirmForgotPassword`
- `ConfirmSignUp`
- `ForgotPassword`
- `InitiateAuth`
- `ResendConfirmationCode`
- `RespondToAuthChallenge`
- `SignUp`

To pass metadata about your user's session to your Amazon Pinpoint campaign, include an `AnalyticsEndpointId` value in the `AnalyticsMetadata` parameter of your API request. For

a JavaScript example, see [Why aren't my Amazon Cognito user pool analytics appearing on my Amazon Pinpoint dashboard?](#) in the [AWS Knowledge Center](#).

Email settings for Amazon Cognito user pools

Certain events in your application can 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 configuration](#) that is built into the Amazon Cognito service.
- [Your Amazon Simple Email Service \(Amazon SES\) configuration](#).

You can change your delivery option after you create your user pool.

Amazon Cognito sends email messages to your users with either a code that they can enter or a URL link that they can select. The following table shows the events that can generate an email message.

Message options

Activity	API operation	Delivery options	Format options	Customizable	Message template
Forgot password	ForgotPassword , AdminResetUserPassword	Email, SMS	code	No	N/A
Invitation	AdminCreateUser	Email, SMS	code	Yes	Invitation message
Self-registration	SignUp , ResendConfirmationCode	Email, SMS	code, link	Yes	Verification message
Email address	UpdateUserAttributes ,	Email, SMS	code	Yes	Verification message

Activity	API operation	Delivery options	Format options	Customizable	Message template
or phone number verification	AdminUpdateUserAttributes , GetUserAttributeVerificationCode				
Multi-factor authentication (MFA)	AdminInitiateAuth , InitiateAuth	Email ¹ , SMS, authenticator app	code	Yes ²	MFA message

¹ Requires advanced security features and [Amazon SES email configuration](#).

² For SMS and email messages.

Amazon SES charges for email messages. For more information, see [Amazon SES pricing](#).

To learn more about email MFA, see [SMS and email message MFA](#).

Default email configuration

Amazon Cognito can use its default email configuration to handle email deliveries for you. When you use the default option, Amazon Cognito limits the number of emails it sends each day for your user pool. For information on service limits, see [Quotas in Amazon Cognito](#). For typical production environments, the default email limit is below the required delivery volume. To enable a higher delivery volume, you can use your Amazon SES email configuration.

When you use the default configuration, you use Amazon SES resources that are managed by AWS to send email messages. Amazon SES adds email addresses that return a [hard bounce](#) to an [account-level suppression list](#) or a [global suppression list](#). If an undeliverable email address becomes deliverable later, you can't control its removal from the suppression list while your user pool is configured to use the default configuration. An email address can remain on the AWS-managed suppression list indefinitely. To manage undeliverable email addresses, use your Amazon SES email configuration with an account-level suppression list, as described in the next section.

When you use the default email configuration, you can use either of the following email addresses as the FROM address:

- The default email address, *no-reply@verificationemail.com*.
- A custom email address. Before you can use your own email address, you must verify it with Amazon SES and grant Amazon Cognito permission to use this address.

Amazon SES email configuration

Your application might require a higher delivery volume than what is available with the default option. To increase the possible delivery volume, use your Amazon SES resources with your user pool to email your users. You can also [monitor your email sending activity](#) when you send email messages with your own Amazon SES configuration.

Before you can use your Amazon SES configuration, you must verify one or more email addresses, or a domain, with Amazon SES. Use a verified email address, or an address from a verified domain, as the FROM email address that you assign to your user pool. When Amazon Cognito sends email to a user, it calls Amazon SES for you and uses your email address.

When you use your Amazon SES configuration, the following conditions apply:

- 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.
- You can manage your messages to undeliverable email addresses with an account-level suppression list in Amazon SES that overrides the [global suppression list](#). When you use an account-level suppression list, email message bounces affect the reputation of your account as a sender. For more information, see [Using the Amazon SES account-level suppression list](#) in the Amazon Simple Email Service Developer Guide.

Amazon SES email configuration Regions

The AWS Region where you create a user pool will have one of three requirements for the configuration of email messages with Amazon SES. You might send email messages from Amazon SES in the same Region as your user pool, several Regions including the same Region, or one or more remote Regions. For best performance, send email messages with a Amazon SES verified identity in the same Region as your user pool when you have the option.

Categories of Region requirements for Amazon SES verified identities

In-Region only

Your user pools can send email messages with verified identities in the same AWS Region as the user pool. In the default email configuration without a custom FROM email address, Amazon Cognito uses a no-reply@verificationemail.com verified identity in the same Region.

Backwards compatible

Your user pools can send email messages with verified identities in the same AWS Region or in one of the following alternate Regions:

- US East (N. Virginia)
- US West (Oregon)
- Europe (Ireland)

This feature supports continuity for user pool resources that you might have created to match Amazon Cognito requirements when the service launched. User pools from that period could only send email messages with verified identities in a limited number of AWS Regions. In the default email configuration without a custom FROM email address, Amazon Cognito uses a no-reply@verificationemail.com verified identity in the same Region.

Alternate Region

Your user pools can send email messages with verified identities in an alternate AWS Region that is outside of the user pool Region. This configuration occurs when Amazon SES isn't available in a Region where Amazon Cognito is available.

The Amazon SES sending authorization policy for your verified identity in the alternate Region must trust the Amazon Cognito service principal of the originating Region. For more information, see [To grant permissions to use the default email configuration](#).

In some of these Regions, Amazon Cognito splits email messages between two alternate Regions for the default email configuration of COGNITO_DEFAULT. In these cases, to use a custom FROM email address, the Amazon SES sending authorization policy for your verified identity in each alternate Region must trust the Amazon Cognito service principal of the originating Region. For more information, see [To grant permissions to use the default email configuration](#). With the Amazon SES email configuration of DEVELOPER in these Regions, you must use a verified identity in the *first* listed Region and configure it to trust the Amazon Cognito service principal in the user pool Region. For example, in a user pool in Middle East (UAE), configure a verified identity in Europe (Frankfurt) to trust cognito-idp.me-

central-1.amazonaws.com. In the default email configuration without a custom FROM email address, Amazon Cognito uses a no-reply@verificationemail.com verified identity in each Region.

Note

Under the following combination of conditions, you must specify the SourceArn parameter of [EmailConfiguration](#) with a wildcard in the Region element, in the format `arn:${Partition}:ses:*:${Account}:identity/${IdentityName}`. This permits your user pool to send email messages with identical verified identities in your AWS account in both AWS Regions.

- Your EmailSendingAccount is COGNITO_DEFAULT.
- You want to use a custom FROM address.
- Your user pool sends emails in an **Alternate Region**.
- Your user pool has a *second¹* **Alternate Region** specified in the table of [Amazon SES supported Regions](#) that follows.

If you create a user pool programmatically—with an AWS SDK, the Amazon Cognito API or CLI, the AWS CDK, or AWS CloudFormation—your user pool sends email messages with the Amazon SES identity that the SourceArn parameter of [EmailConfiguration](#) specifies for your user pool. The Amazon SES identity must occupy a supported AWS Region. If your EmailSendingAccount is COGNITO_DEFAULT and you don't specify a SourceArn parameter, Amazon Cognito sends email messages from no-reply@verificationemail.com using resources in the Region where you created your user pool.

The following table shows the AWS Regions where you can use Amazon SES identities with Amazon Cognito.

User pool Region	Region option	Amazon SES supported Regions
US East (N. Virginia)	Backwards compatible	US East (N. Virginia), US West (Oregon), Europe (Ireland)

User pool Region	Region option	Amazon SES supported Regions
US East (Ohio)	Backwards compatible	US East (Ohio), US East (N. Virginia), US West (Oregon), Europe (Ireland)
US West (N. California)	In-Region only	US West (N. California)
US West (Oregon)	Backwards compatible	US East (N. Virginia), US West (Oregon), Europe (Ireland)
Canada (Central)	Backwards compatible	Canada (Central), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Canada West (Calgary)	Alternate Region	Canada (Central), US West (N. California) ¹
Asia Pacific (Tokyo)	Backwards compatible	Asia Pacific (Tokyo), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Hong Kong)	Alternate Region	Asia Pacific (Singapore), Asia Pacific (Tokyo) ¹
Asia Pacific (Seoul)	Backwards compatible	Asia Pacific (Seoul), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Mumbai)	Backwards compatible	Asia Pacific (Mumbai), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Hyderabad)	Alternate Region	Asia Pacific (Mumbai), Asia Pacific (Singapore) ¹

User pool Region	Region option	Amazon SES supported Regions
Asia Pacific (Singapore)	Backwards compatible	Asia Pacific (Singapore), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Sydney)	Backwards compatible	Asia Pacific (Sydney), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Osaka)	In-Region only	Asia Pacific (Osaka)
Asia Pacific (Jakarta)	In-Region only	Asia Pacific (Jakarta)
Asia Pacific (Melbourne)	Alternate Region	Asia Pacific (Sydney), Asia Pacific (Singapore) ¹
Europe (Ireland)	Backwards compatible	US East (N. Virginia), US West (Oregon), Europe (Ireland)
Europe (London)	Backwards compatible	Europe (London), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Europe (Paris)	In-Region only	Europe (Paris)
Europe (Frankfurt)	Backwards compatible	Europe (Frankfurt), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Europe (Zurich)	Alternate Region	Europe (Frankfurt), Europe (London) ¹
Europe (Stockholm)	In-Region only	Europe (Stockholm)
Europe (Milan)	In-Region only	Europe (Milan)

User pool Region	Region option	Amazon SES supported Regions
Europe (Spain)	Alternate Region	Europe (Paris), Europe (Stockholm) ¹
Middle East (Bahrain)	In-Region only	Middle East (Bahrain)
Middle East (UAE)	Alternate Region	Europe (Frankfurt), Europe (London) ¹
South America (São Paulo)	In-Region only	South America (São Paulo)
Israel (Tel Aviv)	In-Region only	Israel (Tel Aviv)
Africa (Cape Town)	In-Region only	Africa (Cape Town)

¹ Used in user pools with the default email configuration. Amazon Cognito distributes email messages among verified identities with the same email address in each Region. To use a custom FROM address, configure EmailConfiguration with a SourceArn parameter in the format `arn:${Partition}:ses:*:${Account}:identity/${IdentityName}`.

Configuring email for your user pool

Complete the following steps to configure the email settings for your user pool. Depending on the settings that you use, you might need IAM permissions in Amazon SES, AWS Identity and Access Management (IAM), and Amazon Cognito.

Note

You can't share the resources that you create in these steps across AWS accounts. For example, you can't configure a user pool in one account, and then use it with an Amazon SES email address in a different account. If you use Amazon Cognito in multiple accounts, repeat these steps for each account.

Step 1: Verify your email address or domain with Amazon SES

Before you configure your user pool, you must verify one or more domains or 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 or domain, you confirm that you own it, which helps prevent unauthorized use.

For information on verifying an email address with Amazon SES, see [Verifying an Email Address](#) in the *Amazon Simple Email Service Developer Guide*. For information on verifying a domain with Amazon SES, see [Verifying domains](#).

Step 2: Move your account out of the Amazon SES sandbox

Omit this step if you are using the default Amazon Cognito email configuration.

When you first use Amazon SES in any AWS Region, it places your AWS account in the Amazon SES sandbox for that Region. Amazon SES uses the sandbox to prevent fraud and abuse. If you use 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.

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, don't use your Amazon SES configuration for applications that are in production. In this situation, Amazon Cognito can't send messages to your users' email addresses.

To remove your AWS account from 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 configuration, or your Amazon SES configuration.

To grant permissions to use the default email configuration

Complete this step only if you configure your user pool to **Send email with Cognito** or set `EmailSendingAccount` to `COGNITO_DEFAULT`.

With the default email configuration, your user pool can send email messages with either of the following addresses.

- The default address `no-reply@verificationemail.com`.
- A custom FROM address from your verified email addresses or domains in Amazon SES.

If you use a custom address, Amazon Cognito needs additional permissions to email users from that address. These permissions are granted by a [sending authorization policy](#) for the address or domain in Amazon SES. If you use the Amazon Cognito console to add a custom address to your user pool, the policy is automatically attached to the Amazon SES verified email address. However, if you configure your user pool outside of the console, such as using the AWS CLI or the Amazon Cognito API, you must attach the policy using the [Amazon SES console](#) or the [PutIdentityPolicy](#) API.

 **Note**

You can only configure a FROM address in a verified domain using the AWS CLI or the Amazon Cognito API.

A sending authorization policy allows or denies access based on the account resources that are using Amazon Cognito to invoke Amazon SES. For more information about resource-based policies, see the [IAM User Guide](#). You can also find example resource-based policies in the [Amazon SES Developer Guide](#).

Example Sending authorization policy

The following example sending authorization policy grants Amazon Cognito a limited ability to use an Amazon SES verified identity. Amazon Cognito can only send email messages when it does so on behalf of both the user pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

Regions with Amazon SES

Your sending authorization policy in the user pool Region or alternate Region must permit the Amazon Cognito service principal to send email messages. Refer to the [Regions table](#) for more information. If your **User pool Region** matches at least one value in **Amazon SES Region**, configure your sending authorization policy with the global service principal in the following example.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "stmnt1234567891234",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "email.cognito-idp.amazonaws.com"  
                ]  
            },  
            "Action": [  
                "SES:SendEmail",  
                "SES:SendRawEmail"  
            ],  
            "Resource": "<your SES identity ARN>",  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceAccount": "<your account number>"  
                },  
                "ArnLike": {  
                    "aws:SourceArn": "<your user pool ARN>"  
                }  
            }  
        }  
    ]  
}
```

Opt-in Regions without Amazon SES

Amazon SES isn't available in all opt-in AWS Regions where Amazon Cognito is available. Middle East (UAE) is an example, and can only send emails with verified identities in Europe (Frankfurt) (eu-central-1). In user pools with the default email configuration, Amazon Cognito also sends email messages with a verified identity in each of two Regions. In the case of Middle East

(UAE), the additional Region is Europe (London). You must update the sending authorization policy in both Regions.

Your sending authorization policy in each of the alternate Regions must permit the Amazon Cognito service principal in the user pool opt-in Region to send email messages. Refer to the [Regions table](#) for more information. If your Region is marked as **Alternate Region**, configure your sending authorization policies with the Regional service principal as in the following example. Replace the example Region identifier *me-central-1* with the required Region ID as needed.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "cognito-idp.me-central-1.amazonaws.com"  
                ]  
            },  
            "Action": [  
                "SES:SendEmail",  
                "SES:SendRawEmail"  
            ],  
            "Resource": "<your SES identity ARN>",  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceAccount": "<your account number>"  
                },  
                "ArnLike": {  
                    "aws:SourceArn": "<your user pool ARN>"  
                }  
            }  
        }  
    ]  
}
```

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.

Amazon Cognito creates your service-linked role with the AWS credentials of the user session that sets the configuration. The IAM permissions of this session must include the `iam:CreateServiceLinkedRole` action. For more information about permissions in IAM, see [Access management for AWS resources](#) 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](#).

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

Note

If your verified identity is an email address, Amazon Cognito sets that email address as the FROM and REPLY-TO email address by default. But, if your verified identity is a domain, you must provide a value for the FROM email address.

Omit this procedure if you want to use the default Amazon Cognito email configuration and address.

To configure your user pool to use a custom email address

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Messaging** menu, locate **Email configuration**, choose **Edit**.
5. On the **Edit email configuration** page, select **Send email from Amazon SES** or **Send email with Amazon Cognito**. You can customize the **SES Region**, **Configuration Set**, and **FROM sender name** only when you choose **Send email from Amazon SES**.
6. To use a custom FROM address, complete the following steps:
 - a. Under **SES Region**, choose the Region that contains your verified email address.
 - b. Under **FROM email address**, choose your email address. Use an email address that you have verified with Amazon SES.
 - c. (Optional) Under **Configuration set**, choose a configuration set for Amazon SES to use. Making and saving this change creates a service-linked role.
 - d. (Optional) Under **FROM sender address**, enter an email address. You can provide only an email address, or an email address and a friendly name in the format Jane Doe <janedoe@example.com>.
 - e. (Optional) Under **REPLY-TO email address**, enter the email address where you want to receive messages that your users send to your FROM address.
7. Choose **Save changes**.

Related Topics

- [Customizing email verification messages](#)
- [Customizing user invitation messages](#)

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 (Amazon SNS) for delivery of SMS text messages. If you are sending a text message through Amazon Cognito or Amazon SNS for the first time, Amazon SNS places you in a sandbox environment. In the sandbox environment, you can test your applications for SMS text messages. In the sandbox, messages can be sent only to verified phone numbers.

Amazon SNS charges for SMS text messages. For more information, see [Amazon SNS pricing](#).

Note

Because of the volume of unsolicited SMS traffic worldwide, some governments impose barriers between the senders and recipients of SMS messages. When you use SMS messages for MFA and user updates, you must take additional steps to ensure that your messages are delivered. You must also monitor SMS-message-related regulations in countries where your users might live and keep your SMS message configuration current. For more information, see [Mobile text messaging \(SMS\)](#) in the Amazon Simple Notification Service Developer Guide.

The use of SMS messages to authenticate and verify users isn't a security best practice. Phone numbers can change owners, and might not reliably represent a *something you have* factor of MFA for your users. Instead, implement TOTP MFA in your app or with your third-party IdP. You can also create additional custom authentication factors with [Custom authentication challenge Lambda triggers](#).

Amazon Cognito sends SMS messages to your users with a code that they can enter. The following table shows the events that can generate an SMS message.

Message options

Activity	API operation	Delivery options	Format options	Customizable	Message template
Forgot password	ForgotPassword , AdminResetUserPassword	Email, SMS	code	No	N/A
Invitation	AdminCreateUser	Email, SMS	code	Yes	Invitation message

Activity	API operation	Delivery options	Format options	Customizable	Message template
Self-registration	SignUp , ResendConfirmationCode	Email, SMS	code, link	Yes	Verification message
Email address or phone number verification	UpdateUserAttributes , AdminUpdateUserAttributes , GetUserAttributes , GetUserAttributeVerificationCode	Email, SMS	code	Yes	Verification message
Multi-factor authentication (MFA)	AdminInitiateAuth , InitiateAuth	SMS, authenticator app	code	Yes ¹	MFA message

¹ For SMS messages.

Setting up SMS messaging for the first time in Amazon Cognito user pools

Amazon Cognito uses Amazon SNS to send SMS messages to your user pools. You can also use a [Custom SMS sender Lambda trigger](#) to use your own resources to send SMS messages. The first time that you set up Amazon SNS to send SMS text messages in a particular AWS Region, Amazon SNS places your AWS account in the SMS sandbox for that Region. Amazon SNS uses the sandbox to prevent fraud and abuse and to meet compliance requirements. When your AWS account is in the sandbox, Amazon SNS imposes some [restrictions](#). For example, you can send text messages to a maximum of 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

1. [Prepare an IAM role that Amazon Cognito can use to send SMS messages with Amazon SNS](#)

2. [Choose the AWS Region for Amazon SNS SMS messages](#)
3. [Obtain an origination identity to send SMS messages to US phone numbers](#)
4. [Confirm that you are in the SMS sandbox](#)
5. [Move your account out of Amazon SNS sandbox](#)
6. [Verify phone numbers for Amazon Cognito in Amazon SNS](#)
7. [Complete user pool setup in Amazon Cognito](#)

Prepare an IAM role that Amazon Cognito can use to send SMS messages with Amazon SNS

When you send an SMS message from your user pool, Amazon Cognito assumes an IAM role in your account. Amazon Cognito uses the sns:Publish permission assigned to that role to send SMS messages to your users. In the Amazon Cognito console, you can set an **IAM role selection** from the **Messaging** menu of your user pool, under **SMS** or make this selection during the user pool creation wizard.

The following example IAM role trust policy grants Amazon Cognito user pools a limited ability to assume the role. Amazon Cognito can only assume the role when it meets the following conditions:

- The assume-role operation is on behalf of the user pool in the aws:SourceArn condition.
- The assume-role operation is on behalf of a user pool in the AWS account set by the aws:SourceAccount condition.
- The assume-role operation includes the external ID in the sts:externalId condition.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "cognito-idp.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole",  
            "Condition": {  
                "StringEquals": {  
                    "sts:ExternalId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",  
                    "aws:SourceAccount": "111122223333"  
                },  
            }  
    ]}
```

```
        "ArnLike": {
            "aws:SourceArn": "arn:aws:cognito-idp:us-
west-2:111122223333:userpool/us-west-2_EXAMPLE"
        }
    }
}
]
```

You can specify an exact [user pool ARN](#) or a wildcard ARN in the value of the `aws:SourceArn` condition. Look up the ARNs of your user pools in the AWS Management Console or with a [DescribeUserPool](#) API request.

To send SMS messages for [multi-factor authentication](#), your IAM role trust policy must have an `sts:ExternalId` condition. The value of this condition must match the `ExternalId` property of the [SmsConfiguration](#) of your user pool. When you create an IAM role during the process of user pool creation in the Amazon Cognito console, Amazon Cognito configures the external ID for you in the role and in the user pool settings. This isn't true when you use an existing IAM role.

You must update the user pool `ExternalId` parameter in an [UpdateUserPool](#) API request and update the IAM role trust policy with an `sts:externalId` condition with the same value. To learn how to use the API to update a user pool in a way that preserves the original configuration, see [Updating user pool and app client configuration](#).

For more information about IAM roles and trust policies, see [Roles terms and concepts](#) in the *AWS Identity and Access Management User Guide*.

Choose the AWS Region for Amazon SNS SMS messages

In some AWS Regions, you can choose the Region that contains the Amazon SNS resources that you want to use for Amazon Cognito SMS messages. In any AWS Region where Amazon Cognito is available, except for Asia Pacific (Seoul), you can use Amazon SNS resources in the AWS Region where you created your user pool. To make your SMS messaging faster and more reliable when you have a choice of Regions, use Amazon SNS resources in the same Region as your user pool.

Note

In the AWS Management Console, you can only change the Region for SMS resources after you have switched to the new Amazon Cognito console experience.

Choose a Region for SMS resources in the **Configure message delivery** step of the new user pool wizard. You can also choose **Edit** under **SMS** in the **Messaging** menu of an existing user pool.

At launch, for some AWS Regions, Amazon Cognito sent SMS messages with Amazon SNS resources in an alternate Region. To set your preferred Region, use the `SnsRegion` parameter of the [`SmsConfigurationType`](#) object for your user pool. When you programmatically create an Amazon Cognito user pools resource in an **Amazon Cognito Region** from the following table and you do not provide an `SnsRegion` parameter, your user pool can send SMS messages with Amazon SNS resources in a legacy **Amazon SNS Region**.

Amazon Cognito user pools in the Asia Pacific (Seoul) AWS Region must use your Amazon SNS configuration in the Asia Pacific (Tokyo) Region.

Amazon SNS sets the spending quota for all new accounts at \$1.00 (USD) per month. You might have increased your spend limit in an AWS Region that you use with Amazon Cognito. Before you change the AWS Region for Amazon SNS SMS messages, open a quota increase case in the AWS Support Center to increase your limit in the new Region. For more information, see [Requesting increases to your monthly SMS spending quota for Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide*.

You can send SMS messages for any **Amazon Cognito Region** in the following table with Amazon SNS resources in the corresponding **Amazon SNS Region**.

Amazon Cognito Region	Amazon SNS Region
US East (Ohio)	US East (Ohio), US East (N. Virginia)
US East (N. Virginia)	US East (N. Virginia)
US West (N. California)	US West (N. California)
US West (Oregon)	US West (Oregon)
Canada (Central)	Canada (Central), US East (N. Virginia)
Canada West (Calgary)	Canada West (Calgary)
Europe (Frankfurt)	Europe (Frankfurt), Europe (Ireland)
Europe (London)	Europe (London), Europe (Ireland)

Amazon Cognito Region	Amazon SNS Region
Europe (Ireland)	Europe (Ireland)
Europe (Paris)	Europe (Paris)
Europe (Stockholm)	Europe (Stockholm)
Europe (Milan)	Europe (Milan)
Europe (Spain)	Europe (Spain)
Europe (Zurich)	Europe (Zurich)
Asia Pacific (Mumbai)	Asia Pacific (Mumbai), Asia Pacific (Singapore)
Asia Pacific (Hyderabad)	Asia Pacific (Hyderabad)
Asia Pacific (Hong Kong)	Asia Pacific (Singapore)
Asia Pacific (Seoul)	Asia Pacific (Tokyo)
Asia Pacific (Singapore)	Asia Pacific (Singapore)
Asia Pacific (Sydney)	Asia Pacific (Sydney)
Asia Pacific (Tokyo)	Asia Pacific (Tokyo)
Asia Pacific (Jakarta)	Asia Pacific (Jakarta)
Asia Pacific (Osaka)	Asia Pacific (Osaka)
Asia Pacific (Melbourne)	Asia Pacific (Melbourne)
Middle East (Bahrain)	Middle East (Bahrain)
Middle East (UAE)	Middle East (UAE)
South America (São Paulo)	South America (São Paulo)
Israel (Tel Aviv)	Israel (Tel Aviv)

Amazon Cognito Region	Amazon SNS Region
Africa (Cape Town)	Africa (Cape Town)

Obtain an origination identity to send SMS messages to US phone numbers

If you plan to send SMS text messages to US phone numbers, you must obtain an origination identity, regardless of whether you build an SMS sandbox testing environment, or a production environment.

Starting June 1, 2021, US carriers require an origination identity to send messages to US phone numbers. If you don't 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*.

- US East (Ohio)
- Europe (Stockholm)
- Europe (Paris)
- Europe (Milan)
- Middle East (Bahrain)
- South America (São Paulo)
- US West (N. California)

When you have more than one origination identity in the same AWS Region, Amazon SNS chooses an origination identity type in the following order of priority: short code, 10DLC, toll-free number. You can't change this priority. For more information, see [Amazon SNS FAQs](#).

Confirm that you are in the SMS sandbox

Use the following procedure to confirm that you are in the SMS sandbox. Repeat for each AWS Region where you have production Amazon Cognito user pools.

Review SMS sandbox status in the Amazon Cognito console

To confirm that you are in the SMS sandbox

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Messaging** menu.
5. In the **SMS configuration** section, expand **Move to Amazon SNS production environment**. If your account is in the SMS sandbox, you will see the following message:

You are currently in the SMS Sandbox and cannot send SMS messages to unverified numbers.

If you don't see this message, then someone has set up SMS messages in your account already. Skip to [Complete user pool setup in Amazon Cognito](#).

6. Choose the [Amazon SNS](#) link in the message. This opens the Amazon SNS console in a new tab.
7. Verify that you are in the sandbox environment. The console message indicates your sandbox status and AWS Region, as follows:

This account is in the SMS sandbox in US East (N. Virginia).

Move your account out of Amazon SNS sandbox

If you are testing your app and you only need to send SMS messages to phone numbers that your administrators can verify, skip this step.

To use your app in production, move your account out of the SMS sandbox and into production. After you have configured an origination identity in the AWS Region that contains the Amazon SNS resources that you want Amazon Cognito to use, you can verify US phone numbers while your AWS account remains in the SMS sandbox. When your Amazon SNS environment is in production, you don't have to verify user phone numbers in Amazon SNS to send SMS messages to your users.

For detailed instructions, see [Moving Out of the SMS Sandbox](#) in the *Amazon Simple Notification Service Developer Guide*.

Verify phone numbers for Amazon Cognito in Amazon SNS

If you have moved your account out of the SMS sandbox, skip this step.

When you are in the SMS sandbox, you can send messages to any phone number that you have verified with Amazon SNS.

To verify a phone number, do the following:

1. Add a **Sandbox destination phone number** in the **Text messaging (SMS)** section of the Amazon SNS console.
2. Receive an SMS message with a code at the phone number that you provided.
3. Enter the **Verification code** from the SMS message in the Amazon SNS console.

For detailed instructions, see [Adding and verifying phone numbers in the SMS sandbox](#) in the *Amazon Simple Notification Service Developer Guide*.

Note

Amazon SNS limits the number of destination phone numbers that you can verify while you are in the SMS sandbox. See [SMS sandbox](#) in the *Amazon Simple Notification Service Developer Guide*.

Complete user pool setup in Amazon Cognito

Return to the browser tab where you were creating or [editing](#) your user pool. Complete the procedure. When you have successfully added SMS configuration to your user pool, Amazon Cognito sends a test message to an internal phone number to verify that your configuration works. Amazon SNS charges for each test SMS message.

Using Amazon Cognito user pools security features

You might want to secure your application against network intrusion, password guessing, user impersonation, and malicious sign-up and sign-in. Your configuration of Amazon Cognito user pools security features can be a key component in your security architecture. The security of your application is *Customer responsibility "Security in the cloud"* as described in the AWS [Shared Responsibility Model](#). The tools in this chapter contribute to the ability of your application security design to be in line with these goals.

An important decision that you must make when you configure your user pool is whether to permit public sign-up and sign-in. Some user pool option like confidential clients, administrative creation and confirmation of users, and user pools without a domain, are subject to a smaller degree to attacks over the internet. However, a common use case is public clients that accept sign-up from anyone on the internet and send all operations directly to your user pool. In any configuration, but especially in the case of these public configurations, we recommend that you plan and deploy your user pool with security features in mind. Insufficient security can also affect your AWS bill when unwanted sources create new active users or attempt to exploit existing users.

MFA and threat protection apply to [local users](#). Third-party IdPs are responsible for the security posture of [federated users](#).

User pools security features

Multi-factor authentication (MFA)

Request a code that your user pool send by email (with the Essentials or Plus feature plan) or SMS message, or from an authenticator app, to confirm user pool sign-in.

Threat protection

Monitor sign-in for indicators of risk and apply MFA or block sign-in. Add custom claims and scopes to access tokens. Send MFA codes by email.

AWS WAF web ACLs

Inspect incoming traffic to your [user pool endpoints and authentication API](#) for unwanted activity at the network and application layers.

Case sensitivity

Prevent creation of users whose email address or preferred username is identical to another user except for character case.

Deletion protection

Prevent automated systems from accidentally deleting your user pools. Require additional confirmation of user pool deletion in the AWS Management Console.

User existence errors

Guard against disclosure of existing usernames and aliases in your user pool. Return a generic error in response to unsuccessful authentication, whether the username is valid or not.

Topics

- [Adding MFA to a user pool](#)
- [Advanced security with threat protection](#)
- [Associating an AWS WAF web ACL with a user pool](#)
- [User pool case sensitivity](#)
- [User pool deletion protection](#)
- [Managing user existence error responses](#)

Adding MFA to a user pool

MFA adds a *something you have* authentication factor to the initial *something you know* factor that is typically a username and password. You can choose SMS text messages, email messages, or time-based one-time passwords (TOTP) as additional factors to sign in your users.

Multi-factor authentication (MFA) increases security for the [local users](#) in your application. In the case of [federated users](#), Amazon Cognito delegates all authentication processes to the IdP and doesn't offer them additional authentication factors.

Note

The first time that a new user signs in to your app, Amazon Cognito issues OAuth 2.0 tokens, even if your user pool requires MFA. The second authentication factor when your user signs in for the first time is their confirmation of the verification message that Amazon Cognito sends to them. If your user pool requires MFA, Amazon Cognito prompts your user to register an additional sign-in factor to use during each sign-in attempt after the first.

With adaptive authentication, you can configure your user pool to require an additional authentication factor in response to an increased risk level. To add adaptive authentication to your user pool, see [Advanced security with threat protection](#).

When you set MFA to required for a user pool, all users must complete MFA to sign in. To sign in, each user must set up at least one MFA factor. When you set MFA to required, you must include the MFA setup in user onboarding so that your user pool permits them to sign in.

Managed login prompts users to set up MFA when you set MFA to be required. When you set MFA to be optional in your user pool, managed login doesn't prompt users. To work with optional MFA,

you must build an interface in your app that prompts your users to select that they want to set up MFA, then guides them through the API inputs to verify their additional sign-in factor.

Topics

- [Considerations for MFA](#)
- [User MFA preferences](#)
- [Configuring a user pool for multi-factor authentication](#)
- [SMS and email message MFA](#)
- [TOTP software token MFA](#)

Considerations for MFA

Before you set up MFA, consider the following:

- A user's preferred MFA method influences the methods they can use to recover their password. Users whose preferred MFA is by email message can't receive a password-reset code by email. Users whose preferred MFA is by SMS message can't receive a password-reset code by SMS.

Your [password recovery](#) settings must provide an alternative option when users aren't eligible for your preferred password-reset method. For example, your recovery mechanisms might have email as first priority and email MFA might be an option in your user pool. In this case, add SMS-message account recovery as a second option or use administrative API operations to reset passwords for those users.

- When you activate MFA in your user pool and choose **SMS text message** as a second factor, you can send SMS messages to a phone number attribute that you haven't verified in Amazon Cognito. After your user completes SMS MFA, Amazon Cognito sets their `phone_number_verified` attribute to `true`.
- After five unsuccessful attempts to present an MFA code, Amazon Cognito begins the exponential-timeout lockout process described at [An example authentication session](#).
- If your account is in the SMS sandbox in the AWS Region that contains the Amazon Simple Notification Service (Amazon SNS) resources for your user pool, you must verify phone numbers in Amazon SNS before you can send an SMS message. For more information, see [SMS message settings for Amazon Cognito user pools](#).
- To change the MFA status of users in response to detected events with threat protection, activate MFA and set it as optional in the Amazon Cognito user pool console. For more information, see [Advanced security with threat protection](#).

- Email and SMS messages require that your users have email address and phone number attributes respectively. You can set email or phone_number as required attributes in your user pool. In this case, users can't complete sign-up unless they provide a phone number. If you don't set these attributes as required but want to do email or SMS message MFA, you prompt users for their email address or phone number when they sign up. As a best practice, configure your user pool to automatically message users to [verify these attributes](#).

Amazon Cognito counts a phone number or email address as verified if a user has successfully received a temporary code by SMS or email message and returned that code in a [VerifyUserAttribute](#) API request. As an alternative, your team can set phone numbers and mark them as verified with an administrative application that performs [AdminUpdateUserAttributes](#) API requests.

- If you have set MFA to be required and you activated more than one authentication factor, Amazon Cognito prompts new users to select an MFA factor that they want to use. Users must have a phone number to set up SMS message MFA, and an email address to set up email message MFA. If a user doesn't have the attribute defined for any available message-based MFA, Amazon Cognito prompts them to set up TOTP MFA. The prompt to choose an MFA factor (SELECT_MFA_TYPE) and to set up a chosen factor (MFA_SETUP) comes in as a challenge response to [InitiateAuth](#) and [AdminInitiateAuth](#) API operations.

User MFA preferences

Users can set up multiple MFA factors. Only one can be active. You can choose the effective MFA preference for your users in user pool settings or from user prompts. A user pools prompts a user for MFA codes when user pool settings and their own user-level settings meet the following conditions:

1. You set MFA to optional or required in your user pool.
2. The user has a valid email or phone_number attribute, or has set up an authenticator app for TOTP.
3. At least one MFA factor is active.
4. One MFA factor is set as preferred.

User pool settings and their effect on MFA options

The configuration of your user pool influences the MFA methods that users can choose. The following are some user pool settings that influence users' ability to set an MFA preference:

- In the **Multi-factor authentication** configuration in the **Sign-in** menu of the Amazon Cognito console, you can set MFA to optional or required, or turn it off. The API equivalent of this setting is the [MfaConfiguration](#) parameter of `CreateUserPool`, `UpdateUserPool`, and `SetUserPoolMfaConfig`.

Also in the **Multi-factor authentication** configuration, the **MFA methods** setting determines the MFA factors that users can set up. The API equivalent of this setting is the [SetUserPoolMfaConfig](#) operation.

In the **Sign-in** menu, under **User account recovery**, you can configure the way that your user pool sends messages to users who forget their password. A user's preferred MFA method can't have the same delivery method as the highest-priority **Recovery message delivery method** in your user pool. The API equivalent of this configuration is the [AccountRecoverySetting](#) parameter of `CreateUserPool` and `UpdateUserPool`.

For example, users can't set email MFA as preferred when your recovery option is **Email only** or **Email if available, otherwise SMS**. Change the delivery method to **SMS only** or **SMS if available, otherwise email**.

- If you set only one MFA method as available, you don't need to manage user MFA preferences.
- An active SMS configuration automatically makes SMS messages an available MFA method in your user pool.

An active [email configuration](#) with your own Amazon SES resources in a user pool, and the Essentials or Plus feature plan, automatically makes email messages an available MFA method in your user pool.

- When you set MFA to required in a user pool, users can't enable or disable any MFA methods. You can only set a preferred method.
- When you set MFA to optional in a user pool, managed login doesn't prompt users to set up MFA, but it does prompt users for an MFA code when they have a preferred MFA method.
- When you activate [threat protection](#) and configure adaptive-authentication responses in full-function mode, MFA must be optional in your user pool. One of the response options with adaptive authentication is to require MFA for a user whose sign-in attempt is evaluated to contain a level of risk.

The **Required attributes** setting in the **Sign-up** menu of the console determines whether users must provide an email address or phone number to sign up in your application. Email and SMS messages become eligible MFA factors when a user has the corresponding attribute. The [Schema](#) parameter of `CreateUserPool` sets attributes as required.

- When you set MFA to required in a user pool and a user signs in with managed login, Amazon Cognito prompts them to select an MFA method from the available methods for your user pool. Managed login handles the collection of an email address or phone number and the setup of TOTP.

API operations for configuring MFA preferences

You can configure MFA preferences for users in a self-service model with access-token authorization, or in an administrator-managed model with administrative API operations. These operations enable or disable MFA methods and set one of multiple methods as the preferred option. After your user has set an MFA preference, Amazon Cognito prompts them at sign-in to provide a code from their preferred MFA method. Users who have not set a preference receive a prompt to choose a preferred method in a `SELECT_MFA_TYPE` challenge.

- In a user self-service model or public application, [SetUserMfaPreference](#), authorized with a signed-in user's access token, sets MFA configuration.
- In an administrator-managed or confidential application, [AdminSetUserPreference](#), authorized with administrative AWS credentials, sets MFA configuration.

You can also set user MFA preferences from the **Users** menu of the Amazon Cognito console. For more information about the public and confidential authentication models in the Amazon Cognito user pools API, see [Understanding API, OIDC, and managed login pages authentication](#).

Configuring a user pool for multi-factor authentication

You can configure MFA in the Amazon Cognito console.

To configure MFA in the Amazon Cognito console

- Sign in to the [Amazon Cognito console](#).
- Choose **User Pools**.
- Choose an existing user pool from the list, or [create a user pool](#).

4. Choose the **Sign-in** menu. Locate **Multi-factor authentication** and choose **Edit**
5. Choose the **MFA enforcement** method that you want to use with your user pool.

Edit multi-factor authentication (MFA) Info

Amazon Cognito provides your app users with additional authentication factors using SMS messages and time-based one-time passwords (TOTP).

Multi-factor authentication

Configure secure access to your app by enforcing multi-factor authentication (MFA) during the user sign-in process. MFA settings are applied to all app clients.

MFA enforcement Info

- Require MFA - Recommended**
Users must provide an additional authentication factor when signing in.
- Optional MFA**
Users can sign in with a single authentication factor, and can choose to add additional authentication factors.
- No MFA**
Users can only sign in with a single authentication factor. This is the least secure option.

MFA methods Info

Choose the MFA methods that are allowed in your user pool. TOTP-based MFA offers a higher level of security. Recipient message and data rates apply.

- Authenticator apps**
Users can authenticate with a TOTP from an authenticator app such as Authy or Google Authenticator.
- SMS message**
Users can authenticate with a code sent by SMS message to a verified phone number. SMS messages are charged separately by Amazon SNS. [Learn more about pricing](#) This option must be selected because SMS is configured.

[Cancel](#) [Save changes](#)

- a. **Require MFA.** All users in your user pool must sign in with an additional SMS code, or a time-based one-time password (TOTP) factor.
 - b. **Optional MFA** - You can give your users the option to register an additional sign-in factor but still permit users who haven't configured MFA to sign in. If you use adaptive authentication, choose this option. For more information about adaptive authentication, see [Advanced security with threat protection](#).
 - c. **No MFA.** Your users can't register an additional sign-in factor.
6. Choose the **MFA methods** that you support in your app. You can set **Email message**, **SMS message** or TOTP-generating **Authenticator apps** as a second factor.

7. If you use SMS text messages as a second factor and you haven't configured an IAM role to use with Amazon Simple Notification Service (Amazon SNS) for SMS messages, create one in the console. In the **Messaging** menu for your user pool, locate **SMS** and choose **Edit**. You can also use an existing role that allows Amazon Cognito to send SMS messages to your users for you. For more information, see [IAM Roles](#).
8. Choose **Save changes**.

SMS and email message MFA

SMS and email MFA messages confirm that users have access to a message destination before they can sign in. They confirm that they not only have access to a password, but to the SMS messages or the email inbox of the original user. Amazon Cognito requests that users provide a short code that your user pool sent after they successfully provide a username and password.

SMS and email message MFA require no additional configuration after your user adds an email address or phone number to their profile. Amazon Cognito can send messages to unverified email addresses and phone numbers. When a user completes their first MFA, Amazon Cognito marks their email address or phone number as verified.

MFA authentication begins when a user with MFA enters their username and password in your application. Your application submits these initial parameters in an SDK method that invokes an [InitiateAuth](#) or [AdminInitiateAuth](#) API request. The **ChallengeParameters** in the API response includes a **CODE_DELIVERY_DESTINATION** value that indicates where the authorization code was sent. In your application, display a form that prompts the user to check their phone and includes an input element for the code. When they enter their code, submit it in a challenge-response API request to complete the sign-in process.

After a user with MFA signs in with username and password in the [managed login](#) pages, they're automatically prompted for the MFA code.

User pools send SMS messages for MFA and other Amazon Cognito notifications with Amazon Simple Notification Service (Amazon SNS) resources in your AWS account. Similarly, users pools send email messages with Amazon Simple Email Service (Amazon SES) resources in your account. These linked services incur their own costs on your AWS bill for message delivery. They also have additional requirements for sending messages at production volumes. See the following links for more information:

- [SMS message settings for Amazon Cognito user pools](#)

- [Worldwide SMS Pricing](#)
- [Email settings for Amazon Cognito user pools](#)
- [Amazon SES pricing](#)

Considerations for SMS and email message MFA

- To permit users to sign in with email MFA, your user pool must have the following configuration options:
 1. You have the Plus or Essentials feature plan in your user pool. For more information, see [User pool feature plans](#).
 2. Your user pool sends email messages with your own Amazon SES resources. For more information, see [Amazon SES email configuration](#).
- The MFA code is valid for the **Authentication flow session duration** that you set for your app client.

Set the duration of an authentication flow session in the Amazon Cognito console in the **App clients** menu when you **Edit** your app client. You can also set the authentication flow session duration in a `CreateUserPoolClient` or `UpdateUserPoolClient` API request. For more information, see [An example authentication session](#).

- When a user successfully provides a code from an SMS or email message that Amazon Cognito sent to an unverified phone number or email address, Amazon Cognito marks the corresponding attribute as verified.
- A user can't use their access token to change the value of a phone number or email address that's associated with MFA. Your team must change these values with administrator AWS credentials in [AdminUpdateUserAttributes](#) API requests.
- For a user to make a self-service change to the value of a phone number or email address that's associated with MFA, they must sign in and authorize the request with an access token. If they can't access their current phone number or email address, they can't sign in. Your team must change these values with administrator AWS credentials in [AdminUpdateUserAttributes](#) API requests.
- After you [configure SMS](#) in your user pool, you can't disable SMS messages as an available MFA factor.

TOTP software token MFA

When you set up TOTP software token MFA in your user pool, your user signs in with a username and password, then uses a TOTP to complete authentication. After your user sets and verifies a username and password, they can activate a TOTP software token for MFA. If your app uses the Amazon Cognito managed login to sign in users, your user submits their username and password, and then submits the TOTP password on an additional sign-in page.

You can activate TOTP MFA for your user pool in the Amazon Cognito console, or you can use Amazon Cognito API operations. At the user pool level, you can call [SetUserPoolMfaConfig](#) to configure MFA and enable TOTP MFA.

Note

If you haven't activated TOTP software token MFA for the user pool, Amazon Cognito can't use the token to associate or verify users. In this case, users receive a `SoftwareTokenMFANotFoundException` exception with the description Software Token MFA has not been enabled by the `userPool`. If you deactivate software token MFA for the user pool later, users who previously associated and verified a TOTP token can continue to use it for MFA.

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.

When you configure your user pool to require TOTP MFA and your users sign up for your app in managed login, Amazon Cognito automates the user process. Amazon Cognito prompts your user to choose an MFA method, displays a QR code to set up their authenticator app, and verifies their MFA registration. In user pools where you have allowed users to choose between SMS and TOTP MFA, Amazon Cognito also presents your user with a choice of method.

Important

When you have an AWS WAF web ACL associated with a user pool, and a rule in your web ACL presents a CAPTCHA, this can cause an unrecoverable error in managed login TOTP registration. To create a rule that has a CAPTCHA action and doesn't affect managed login TOTP, see [Configuring your AWS WAF web ACL for managed login TOTP MFA](#). For more

information about AWS WAF web ACLs and Amazon Cognito, see [Associating an AWS WAF web ACL with a user pool](#).

To implement TOTP MFA in a custom-built UI with an AWS SDK and the [Amazon Cognito user pools API](#), see [Configuring MFA for a user in the Amazon Cognito user pools API](#).

To add MFA to your user pool, see [Adding MFA to a user pool](#).

TOTP MFA considerations and limitations

1. Amazon Cognito supports software token MFA through an authenticator app that generates TOTP codes. Amazon Cognito doesn't support hardware-based MFA.
2. When your user pool requires TOTP for a user who has not configured it, your user receives a one-time access token that your app can use to activate TOTP MFA for the user. Subsequent sign-in attempts fail until your user has registered an additional TOTP sign-in factor.
 - A user who signs up in your user pool with the `SignUp` API operation or through managed login receives one-time tokens when the user completes sign-up.
 - After you create a user, and the user sets their initial password, Amazon Cognito issues one-time tokens from managed login to the user. If you set a permanent password for the user, Amazon Cognito issues one-time tokens when the user first signs in.
 - Amazon Cognito doesn't issue one-time tokens to an administrator-created user who signs in with the [InitiateAuth](#) or [AdminInitiateAuth](#) API operations. After your user succeeds in the challenge to set their initial password, or if you set a permanent password for the user, Amazon Cognito immediately challenges the user to set up MFA.
3. If a user in a user pool that requires MFA has already received a one-time access token but hasn't set up TOTP MFA, the user can't sign in with managed login until they have set up MFA. Instead of the access token, you can use the session response value from an `MFA_SETUP` challenge to [InitiateAuth](#) or [AdminInitiateAuth](#) in an [AssociateSoftwareToken](#) request.
4. If your users have set up TOTP, they can use it for MFA, even if you deactivate TOTP for the user pool later.
5. Amazon Cognito only accepts TOTPs from authenticator apps that generate codes with the HMAC-SHA1 hash function. Codes generated with SHA-256 hashing return a `Code mismatch` error.

Configuring MFA for a user in the Amazon Cognito user pools API

When a user first signs in, your app uses their one-time access token to generate the TOTP private key and present it to your user in text or QR code format. Your user configures their authenticator app and provides a TOTP for subsequent sign-in attempts. Your app or managed login presents the TOTP to Amazon Cognito in MFA challenge responses.

Topics

- [Associate the TOTP software token](#)
- [Verify the TOTP token](#)
- [Sign in with TOTP MFA](#)
- [Remove the TOTP token](#)

Associate the TOTP software token

To associate the TOTP token, send your user a secret code that they must validate with a one-time password. Associating the token requires 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. You can authorize AssociateSoftwareToken with either an access token or a session string.
2. Your app presents the user with the private key, or a QR code that you generate from the private key. Your user must enter the key into a TOTP-generating app such as Google Authenticator. You can use [libqrencode](#) to generate a QR code.
3. Your user enters the key, or scans the QR code into a authenticator app such as Google Authenticator, and the app begins generating codes.

Verify the TOTP token

Next, verify the TOTP token. Request sample codes from your user and provide them to the Amazon Cognito service to confirm that the user is successfully generating TOTP codes, as follows.

1. Your app prompts your user for a code to demonstrate that they have set up their authenticator app properly.
2. The user's authenticator app displays a temporary password. The authenticator app bases the password on the secret key you gave to the user.

3. Your user enters their temporary password. Your app passes the temporary password to Amazon Cognito in a [VerifySoftwareToken](#) API request.
4. Amazon Cognito has retained the secret key associated with the user, and generates a TOTP and compares it with the one that your user provided. If they match, VerifySoftwareToken returns a SUCCESS response.
5. Amazon Cognito associates the TOTP factor with the user.
6. If the VerifySoftwareToken operation returns an ERROR response, make sure that the user's clock is correct and that they have not exceeded the maximum number of retries. Amazon Cognito accepts TOTP tokens that are within 30 seconds before or after the attempt, to account for minor clock skew. When you have resolved the issue, try the VerifySoftwareToken operation again.

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 username 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 deactivate their TOTP configuration. Currently, you can't delete a user's TOTP software token. To replace your user's software token, associate and verify a new software token. To deactivate TOTP MFA for a user, call [SetUserMFAPreference](#) to modify your user to use no MFA, or only SMS MFA.

1. Create an interface in your app for users who want to reset MFA. Prompt a user in this interface to enter their password.

2. If Amazon Cognito returns a TOTP MFA challenge, update your user's MFA preference with [SetUserMFAPreference](#).
3. In your app, communicate to your user that they have deactivated MFA and prompt them to sign in again.

Configuring your AWS WAF web ACL for managed login TOTP MFA

When you have an AWS WAF web ACL associated with a user pool, and a rule in your web ACL presents a CAPTCHA, this can cause an unrecoverable error in managed login and managed login TOTP registration. AWS WAF CAPTCHA rules only affect TOTP MFA in managed login and the managed login in this way. SMS MFA is unaffected.

Amazon Cognito displays the following error when your CAPTCHA rule doesn't let a user complete TOTP MFA setup.

Request not allowed due to WAF captcha.

This error results when AWS WAF prompts for a CAPTCHA in response to [AssociateSoftwareToken](#) and [VerifySoftwareToken](#) API requests that your user pool makes in the background. To create a rule that has a CAPTCHA action and doesn't affect TOTP in managed login pages, exclude the x-amzn-cognito-operation-name header values of AssociateSoftwareToken and VerifySoftwareToken from the CAPTCHA action in your rule.

The following screenshot shows an example AWS WAF rule that applies a CAPTCHA action to all requests that don't have a x-amzn-cognito-operation-name header value of AssociateSoftwareToken or VerifySoftwareToken.

If a request matches all the statements (AND)

NOT Statement 1

Field to match

Single header (x-amzn-cognito-operation-name)

Positional constraint

Exactly matches string

Search string

AssociateSoftwareToken

Text transformations

- None (Priority 0)

AND

NOT Statement 2

Field to match

Single header (x-amzn-cognito-operation-name)

Positional constraint

Exactly matches string

Search string

VerifySoftwareToken

Text transformations

- None (Priority 0)

Then

Action

The action to take when a web request matches the rule statement.

For more information about AWS WAF web ACLs and Amazon Cognito, see [Associating an AWS WAF web ACL with a user pool](#).

Advanced security with threat protection

After you create your user pool, you have access to **Threat protection** in the navigation menu in the Amazon Cognito console. You can turn threat protection 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 applying any security mitigations. In audit mode, threat protection publishes metrics to Amazon CloudWatch. You can see metrics after Amazon Cognito generates its first event. See [Viewing threat protection metrics](#).

Threat protection, formerly called *advanced security features*, is a set of monitoring tools for unwanted activity in your user pool, and configuration tools to automatically shut down potentially malicious activity. Threat protection has different configuration options for standard and custom authentication operations. For example, you might want to send a notification to a user with a suspicious custom authentication sign-in, where you have set up additional security factors, but block a user at the same risk level with basic username-password authentication.

Threat protection is available in the Plus feature plan. For more information, see [User pool feature plans](#).

The following user pool options are the components of threat protection.

Compromised credentials

Users reuse passwords for multiple user accounts. The compromised credentials feature of Amazon Cognito compiles data from public leaks of user names and passwords, and compares your users' credentials to lists of leaked credentials. Compromised credentials detection also checks for commonly-guessed passwords. You can check for compromised credentials in username-and-password standard authentication flows in user pools. Amazon Cognito doesn't detect compromised credentials in secure remote password (SRP) or custom authentication.

You can choose the user actions that prompt a check for compromised credentials, and the action that you want Amazon Cognito to take in response. For sign-in, sign-up, and password-change events, Amazon Cognito can **Block sign-in**, or **Allow sign-in**. In both cases, Amazon Cognito generates a user activity log where you can find more information about the event.

Adaptive authentication

Amazon Cognito can review location and device information from your users' sign-in requests and apply an automatic response to secure the user accounts in your user pool against suspicious activity. You can monitor user activity and automate responses to detected risk levels in username-password and SRP, and custom authentication.

When you activate threat protection, Amazon Cognito assigns a risk score to user activity. You can assign an automatic response to suspicious activity: you can **Require MFA**, **Block sign-in**, or just log the activity details and risk score. You can also automatically send email messages that notify your user of the suspicious activity so that they can reset their password or take other self-guided actions.

IP address allowlist and denylist

With Amazon Cognito threat protection in **Full function** mode, you can create IP address **Always block** and **Always allow** exceptions. A session from an IP address on the **Always block** exception list isn't assigned a risk level by adaptive authentication, and can't sign in to your user pool.

Log export

Threat protection logs granular details of users' authentication requests to your user pool. These logs feature threat assessments, user information, and session metadata like location and device. You can create external archives of these logs for retention and analysis. Amazon Cognito user pools export threat protection logs to Amazon S3, CloudWatch Logs, and Amazon Data Firehose. For more information, see [Viewing and exporting user event history](#).

Topics

- [Considerations and limitations for threat protection](#)
- [Turning on threat protection in user pools](#)
- [Threat protection enforcement concepts](#)
- [Threat protection for standard authentication and custom authentication](#)
- [Threat protection prerequisites](#)
- [Setting up threat protection](#)
- [Working with compromised-credentials detection](#)
- [Working with adaptive authentication](#)
- [Viewing threat protection metrics](#)

- [Collecting data for threat protection in applications](#)

Considerations and limitations for threat protection

Threat protection options differ between authentication flows

Amazon Cognito supports both adaptive authentication and compromised-credentials detection with the authentication flows USER_PASSWORD_AUTH and ADMIN_USER_PASSWORD_AUTH. You can enable only adaptive authentication for USER_SRP_AUTH. You can't use threat protection with federated sign-in.

Always-block IPs contribute to request quotas

Blocked requests from IP addresses on an **Always block** exception list in your user pool contribute to the [request rate quotas](#) for your user pools.

Threat protection doesn't apply rate limits

Some malicious traffic has the characteristic of a high volume of requests, like distributed denial of service (DDoS) attacks. The risk ratings that Amazon Cognito applies to incoming traffic are per-request and don't take request volume into account. Individual requests in a high-volume event might receive a risk score and an automated response for application-layer reasons that aren't related to their role in a volumetric attack. To implement defenses against volumetric attacks in your user pools, add AWS WAF web ACLs. For more information, see [Associating an AWS WAF web ACL with a user pool](#).

Threat protection doesn't affect M2M requests

Client credentials grants are intended for machine-to-machine (M2M) authorization with no connection to user accounts. Threat protection only monitors user accounts and passwords in your user pool. To implement security features with your M2M activity, consider the capabilities of AWS WAF for monitoring request rates and content. For more information, see [Associating an AWS WAF web ACL with a user pool](#).

Turning on threat protection in user pools

Amazon Cognito user pools console

To activate threat protection for a user pool

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.

2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. If you haven't already, activate the Plus feature plan from the **Settings** menu.
5. Choose the **Threat protection** menu and select **Activate**.
6. Choose **Save changes**.

API

Set your feature plan to Plus in a [CreateUserPool](#) or [UpdateUserPool](#) API request. The following partial example request body sets threat protection to full-function mode. For a complete example request, see [Examples](#).

```
"UserPoolAddOns": {  
    "AdvancedSecurityMode": "ENFORCED"  
}
```

Threat protection is the collective term for the features that monitor user operations for signs of account takeover and automatically respond to secure affected user accounts. You can apply threat protection settings to users when they sign in with standard and custom authentication flows.

Threat protection [generates logs](#) that detail users' sign-in, sign-out, and other activity. You can export these logs to a third-party system. For more information, see [Viewing and exporting user event history](#).

Threat protection enforcement concepts

Threat protection starts out in an *audit-only* mode where your user pool monitors user activity, assigns risk levels, and generates logs. As a best practice, run in audit-only mode for two weeks or more before you enable *full-function mode*. Full-function mode includes a set of automatic reactions to detected risky activity and compromised passwords. With audit-only mode, you can monitor the threat assessments that Amazon Cognito is performing. You can also [provide feedback](#) that trains the feature on false positives and negatives.

You can configure threat protection enforcement at the user pool level to cover all app clients in the user pool, and at the level of individual app clients. App client threat-protection configurations override the user pool configuration. To configure threat protection for an app client, navigate to the app client settings from the **App clients** menu of your user pool in the Amazon Cognito

console. There, you can **Use client-level settings** and configure enforcement exclusive to the app client.

Additionally, you can configure threat protection separately for standard and custom authentication types.

Threat protection for standard authentication and custom authentication

The ways that you can configure threat protection depend on the type of authentication you're doing in your user pool and app clients. Each of the following types of authentication can have their own enforcement mode and automated responses.

Standard authentication

Standard authentication is user sign-in, sign-out and password management with username-password flows and in managed login. Amazon Cognito threat protection monitors operations for indicators of risk when they sign in with managed login or use the following API AuthFlow parameters:

[InitiateAuth](#)

USER_PASSWORD_AUTH, USER_SRP_AUTH. The compromised credentials feature doesn't have access to passwords in USER_SRP_AUTH sign-in, and doesn't monitor or act on events with this flow.

[AdminInitiateAuth](#)

ADMIN_USER_PASSWORD_AUTH, USER_SRP_AUTH. The compromised credentials feature doesn't have access to passwords in USER_SRP_AUTH sign-in, and doesn't monitor or act on events with this flow.

You can set the **Enforcement mode** for standard authentication to **Audit only** or **Full function**. To disable threat monitoring for standard authentication, set threat protection to **No enforcement**.

Custom authentication

Custom authentication is user sign-in with [custom challenge Lambda triggers](#). You can't do custom authentication in managed login. Amazon Cognito threat protection monitors operations for indicators of risk when they sign in with the API AuthFlow parameter CUSTOM_AUTH of InitiateAuth and AdminInitiateAuth.

You can set the **Enforcement mode** for custom authentication to **Audit only**, **Full function**, or **No enforcement**. The **No enforcement** option disables threat monitoring for custom authentication without affecting other threat protection features.

Threat protection prerequisites

Before you begin, you need the following:

- A user pool with an app client. For more information, see [Getting started with user pools](#).
- 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 MFA to a user pool](#).
- If you're using email notifications, go to the [Amazon SES console](#) to configure and verify an email address or domain to use with your email notifications. For more information about Amazon SES, see [Verifying Identities in Amazon SES](#).

Setting up threat protection

Follow these instructions to set up user pool threat protection.

Note

To set up a different threat protection configuration for an app client in the Amazon Cognito user pools console, select the app client from the **App clients** menu and choose **Use client-level settings**.

AWS Management Console

To configure threat protection for a user pool

1. Go to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Threat protection** menu and select **Activate**.
5. Choose the threat protection method that you want to configure: **Standard and custom authentication**. You can set different enforcement modes for custom and standard

authentication, but they share the configuration of automated responses in **Full function** mode.

6. Select **Edit**.
7. Choose an **Enforcement mode**. To start responding to detected risks immediately, select **Full function** and configure the automated responses for compromised credentials and adaptive authentication. To gather information in user-level logs and in CloudWatch, select **Audit only**.

We recommend that you keep threat protection in audit mode for two weeks before enabling actions. During this time, Amazon Cognito can learn the usage patterns of your app users and you can provide event feedback to adjust responses.

8. If you selected **Audit only**, choose **Save changes**. If you selected **Full function**:
 - a. Select whether you will take **Custom** action or use or **Cognito defaults** to respond to suspected **Compromised credentials**. **Cognito defaults** are:
 - i. Detect compromised credentials on **Sign-in**, **Sign-up**, and **Password change**.
 - ii. Respond to compromised credentials with the action **Block sign-in**.
 - b. If you selected **Custom** actions for **Compromised credentials**, choose the user pool actions that Amazon Cognito will use for **Event detection** and the **Compromised credentials responses** that you would like Amazon Cognito to take. You can **Block sign-in** or **Allow sign-in** with suspected compromised credentials.
 - c. Choose how to respond to malicious sign-in attempts under **Adaptive authentication**. Select whether you will take **Custom** action or use or **Cognito defaults** to respond to suspected malicious activity. When you select **Cognito defaults**, Amazon Cognito blocks sign-in at all risk levels and does not notify the user.
 - d. If you selected **Custom** actions for **Adaptive authentication**, choose the **Automatic risk response** actions that Amazon Cognito will take in response to detected risks based on severity level. When you assign a response to a level of risk, you can't assign a less-restrictive response to a higher level of risk. You can assign the following responses to risk levels:
 - i. **Allow sign-in** - Take no preventative action.
 - ii. **Optional MFA** - If the user has MFA configured, Amazon Cognito will always require the user to provide an additional SMS or time-based one-time password

- (TOTP) factor when they sign in. If the user does not have MFA configured, they can continue signing in normally.
- iii. **Require MFA** - If the user has MFA configured, Amazon Cognito will always require the user to provide an additional SMS or TOTP factor when they sign in. If the user does not have MFA configured, Amazon Cognito will prompt them to set up MFA. Before you automatically require MFA for your users, configure a mechanism in your app to capture phone numbers for SMS MFA, or to register authenticator apps for TOTP MFA.
 - iv. **Block sign-in** - Prevent the user from signing in.
 - v. **Notify user** - Send an email message to the user with information about the risk that Amazon Cognito detected and the response you have taken. You can customize email message templates for the messages you send.
9. If you chose **Notify user** in the previous step, you can customize your email delivery settings and email message templates for adaptive authentication.
- a. Under **Email configuration**, choose the **SES Region**, **FROM email address**, **FROM sender name**, and **REPLY-TO email address** that you want to use with adaptive authentication. For more information about integrating your user pool email messages with Amazon Simple Email Service, see [Email settings for Amazon Cognito user pools](#).

Adaptive authentication messages

Customize the messages sent to users when adaptive authentication triggers a notification. Adaptive authentication messages use [Amazon SES](#).

Email configuration

Configure the [Amazon SES](#) verified identity used to send adaptive authentication messages. [Learn more](#)

SES Region | [Info](#)

Choose an AWS Region to use with SES in this user pool. For best performance, you should configure SES and your user pool in the same Region.

US East (N. Virginia)

FROM email address | [Info](#)

Choose an email address that you have verified with Amazon SES.



FROM sender name - *optional* | [Info](#)

Enter a friendly name for the email sender in the format "John Stiles <johnstiles@example.com>."

REPLY-TO email address - *optional* | [Info](#)

If you set an invalid reply-to address, sending restrictions may be imposed on your account.

▼ Email templates

Risk detected, sign-in allowed

Email subject

[Reset to default](#)

New sign-in attempt

Email message - Text

[Reset to default](#)

We observed an unrecognized sign-in to your



<!DOCTYPE html>

[Reset to default](#)



- b. Expand **Email templates** to customize adaptive authentication notifications with both HTML and plaintext versions of email messages. To learn more about email message templates, see [Message templates](#).
10. Expand **IP address exceptions** to create an **Always-allow** or an **Always-block** list of IPv4 or IPv6 address ranges that will always be allowed or blocked, regardless of the threat protection risk assessment. Specify the IP address ranges in [CIDR notation](#) (such as 192.168.100.0/24).
11. Choose **Save changes**.

API (user pool)

To set the threat protection configuration for a user pool, send a [SetRiskConfiguration](#) API request that includes a `UserPoolId` parameter, but not a `ClientId` parameter. The following is an example request body for a user pool. This risk configuration takes an escalating series of actions based on the severity of risk and notifies users at all risk levels. It applies a compromised-credentials block to sign-up operations.

To enforce this configuration, you must set `AdvancedSecurityMode` to `ENFORCED` in a separate [CreateUserPool](#) or [UpdateUserPool](#) API request. For more information about the placeholder templates like `{username}` in this example, see [Configuring verification and invitation messages](#).

```
{  
    "AccountTakeoverRiskConfiguration": {  
        "Actions": {  
            "HighAction": {  
                "EventAction": "MFA_REQUIRED",  
                "Notify": true  
            },  
            "LowAction": {  
                "EventAction": "NO_ACTION",  
                "Notify": true  
            },  
            "MediumAction": {  
                "EventAction": "MFA_IF_CONFIGURED",  
                "Notify": true  
            }  
        },  
        "NotifyConfiguration": {  
            "BlockEmail": {  
                "Subject": "You have been blocked for suspicious activity",  
                "TextBody": "We blocked {username} at {login-time} from {ip-address}."  
            },  
            "From": "admin@example.com",  
            "MfaEmail": {  
                "Subject": "Suspicious activity detected, MFA required",  
                "TextBody": "Unexpected sign-in from {username} on device {device-name}.  
You must use MFA."  
            },  
            "NoActionEmail": {  
                "Subject": "Suspicious activity detected, secure your user account",  
            }  
        }  
    }  
}
```

```
        "TextBody": "We noticed suspicious sign-in activity by {username} from {city}, {country} at {login-time}. If this was not you, reset your password."
    },
    "ReplyTo": "admin@example.com",
    "SourceArn": "arn:aws:ses:us-west-2:123456789012:identity/admin@example.com"
}
},
"CompromisedCredentialsRiskConfiguration": {
    "Actions": {
        "EventAction": "BLOCK"
    },
    "EventFilter": [ "SIGN_UP" ]
},
"RiskExceptionConfiguration": {
    "BlockedIPRangeList": [ "192.0.2.0/24", "198.51.100.0/24" ],
    "SkippedIPRangeList": [ "203.0.113.0/24" ]
},
"UserPoolId": "us-west-2_EXAMPLE"
}
```

API (app client)

To set the threat protection configuration for an app client, send a [SetRiskConfiguration](#) API request that includes a `UserPoolId` parameter and a `ClientId` parameter. The following is an example request body for an app client. This risk configuration is more severe than the user pool configuration, blocking high-risk entries. It also applies compromised-credentials blocks to sign-up, sign-in, and password-reset operations.

To enforce this configuration, you must set `AdvancedSecurityMode` to `ENFORCED` in a separate [CreateUserPool](#) or [UpdateUserPool](#) API request. For more information about the placeholder templates like `{username}` in this example, see [Configuring verification and invitation messages](#).

```
{
    "AccountTakeoverRiskConfiguration": {
        "Actions": {
            "HighAction": {
                "EventAction": "BLOCK",
                "Notify": true
            },
            "LowAction": {

```

```
        "EventAction": "NO_ACTION",
        "Notify": true
    },
    "MediumAction": {
        "EventAction": "MFA_REQUIRED",
        "Notify": true
    }
},
"NotifyConfiguration": {
    "BlockEmail": {
        "Subject": "You have been blocked for suspicious activity",
        "TextBody": "We blocked {username} at {login-time} from {ip-address}."
    },
    "From": "admin@example.com",
    "MfaEmail": {
        "Subject": "Suspicious activity detected, MFA required",
        "TextBody": "Unexpected sign-in from {username} on device {device-name}. You must use MFA."
    },
    "NoActionEmail": {
        "Subject": "Suspicious activity detected, secure your user account",
        "TextBody": "We noticed suspicious sign-in activity by {username} from {city}, {country} at {login-time}. If this was not you, reset your password."
    },
    "ReplyTo": "admin@example.com",
    "SourceArn": "arn:aws:ses:us-west-2:123456789012:identity/admin@example.com"
}
},
"ClientId": "1example23456789",
"CompromisedCredentialsRiskConfiguration": {
    "Actions": {
        "EventAction": "BLOCK"
    },
    "EventFilter": [ "SIGN_UP", "SIGN_IN", "PASSWORD_CHANGE" ]
},
"RiskExceptionConfiguration": {
    "BlockedIPRangeList": [ "192.0.2.1/32", "192.0.2.2/32" ],
    "SkippedIPRangeList": [ "192.0.2.3/32", "192.0.2.4/32" ]
},
"UserPoolId": "us-west-2_EXAMPLE"
}
```

Working with compromised-credentials detection

Amazon Cognito can detect if a user's username and password have been compromised elsewhere. This can happen when users reuse credentials at more than one site, or when they use insecure passwords. Amazon Cognito checks *local users* who sign in with username and password, in managed login and with the Amazon Cognito API. A local user exists exclusively in your user pool directory without federation through an external IdP.

From the **Threat protection** menu of the Amazon Cognito console, you can configure **Compromised credentials**. Configure **Event detection** to choose the user events that you want to monitor for compromised credentials. Configure **Compromised credentials responses** to choose whether to allow or block the user if compromised credentials are detected. Amazon Cognito can check for compromised credentials during sign-in, sign-up, and password changes.

When you choose **Allow sign-in**, you can review Amazon CloudWatch Logs to monitor the evaluations that Amazon Cognito makes on user events. For more information, see [Viewing threat protection metrics](#). When you choose **Block sign-in**, Amazon Cognito prevents sign-in by users who use compromised credentials. When Amazon Cognito blocks sign-in for a user, it sets the user's [UserStatus](#) to RESET_REQUIRED. A user with a RESET_REQUIRED status must change their password before they can sign in again.

Note

Currently, Amazon Cognito doesn't check for compromised credentials for sign-in operations with Secure Remote Password (SRP) flow. SRP sends a hashed proof of password during sign-in. Amazon Cognito doesn't have access to passwords internally, so it can only evaluate a password that your client passes to it in plaintext.

Amazon Cognito checks sign-ins that use the [AdminInitiateAuth](#) API with ADMIN_USER_PASSWORD_AUTH flow, and the [InitiateAuth](#) API with USER_PASSWORD_AUTH flow, for compromised credentials.

To add compromised credentials protections to your user pool, see [Advanced security with threat protection](#).

Working with 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 device and user factors that your application provides, and others that Amazon Cognito derives from the request. Some factors that contribute to the risk evaluation by Amazon Cognito are IP address, user agent, and geographical distance from other sign-in attempts. Adaptive authentication can turn on or require multi-factor authentication (MFA) for a user in your user pool when Amazon Cognito detects risk in a user's session, and the user hasn't yet chosen an MFA method. When you activate MFA for a user, they always receive a challenge to provide or set up a second factor during authentication, regardless of how you configured adaptive authentication. From your user's perspective, your app offers to help them set up MFA, and optionally Amazon Cognito prevents them from signing in again until they have configured an additional factor.

Amazon Cognito publishes metrics about sign-in attempts, their risk levels, and failed challenges to Amazon CloudWatch. For more information, see [Viewing threat protection metrics](#).

To add adaptive authentication to your user pool, see [Advanced security with threat protection](#).

Topics

- [Adaptive authentication overview](#)
- [Adding user device and session data to API requests](#)
- [Viewing and exporting user event history](#)
- [Providing event feedback](#)
- [Sending notification messages](#)

Adaptive authentication overview

From the **Threat protection** menu 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. You can assign a global threat protection configuration to all of your app clients, but apply a client-level configuration to individual app clients.

Amazon Cognito adaptive authentication assigns one of the following risk levels to each user session: **High**, **Medium**, **Low**, or **No Risk**.

Consider your options carefully when you change your **Enforcement method** from **Audit-only** to **Full-function**. The automatic responses that you apply to risk levels influence the risk level that Amazon Cognito assigns to subsequent user sessions with the same characteristics. For example,

after you choose to take no action, or **Allow**, user sessions that Amazon Cognito initially evaluates to be high-risk, Amazon Cognito considers similar sessions to have a lower risk.

For each risk level, you can choose from the following options:

Option	Action
Allow	Users can sign in without an additional factor.
Optional MFA	Users who have a second factor configured must complete a second factor challenge to sign in. A phone number for SMS and a TOTP software token are the available second factors. Users without a second factor configured can sign in with only one set of credentials.
Require MFA	Users who have a second factor configured must complete a second factor challenge to sign in. Amazon Cognito blocks sign-in for users who don't have a second factor configured.
Block	Amazon Cognito blocks all sign-in attempts at the designated risk level.

 **Note**

You don't have to verify phone numbers to use them for SMS as a second authentication factor.

Adding user device and session data to API requests

You can collect and pass information about your user's session to Amazon Cognito threat protection when you use the API to sign them up, sign them in, and reset their password. This information includes your user's IP address and a unique device identifier.

You might have an intermediate network device between your users and Amazon Cognito, like a proxy service or an application server. You can collect users' context data and pass it to Amazon Cognito so that adaptive authentication calculates your risk based on the characteristics of the user endpoint, instead of your server or proxy. If your client-side app calls Amazon Cognito API operations directly, adaptive authentication automatically records the source IP address. However, it does not record other device information like the user-agent unless you also collect a device fingerprint.

Generate this data with the Amazon Cognito context data collection library and submit it to Amazon Cognito threat protection with the [ContextData](#) and [UserContextData](#) parameters. The context data collection library is included in the AWS SDKs. For more information, see [Integrating Amazon Cognito authentication and authorization with web and mobile apps](#). You can submit ContextData if you have the Plus feature plan. For more information, see [Setting up threat protection](#).

When you call the following Amazon Cognito authenticated API operations from your application server, pass the IP of the user's device in the ContextData parameter. In addition, pass your server name, server path, and encoded device-fingerprinting data.

- [AdminInitiateAuth](#)
- [AdminRespondToAuthChallenge](#)

When you call Amazon Cognito unauthenticated API operations, you can submit UserContextData to Amazon Cognito threat protection. This data includes a device fingerprint in the EncodedData parameter. You can also submit an IpAddress parameter in your UserContextData if you meet the following conditions:

- Your user pool is on the Plus feature plan. For more information, see [User pool feature plans](#).
- Your app client has a client secret. For more information, see [Application-specific settings with app clients](#).
- You have activated **Accept additional user context data** in your app client. For more information, see [Accepting additional user context data \(AWS Management Console\)](#).

Your app can populate the UserContextData parameter with encoded device-fingerprinting data and the IP address of the user's device in the following Amazon Cognito unauthenticated API operations.

- [InitiateAuth](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [ConfirmSignUp](#)
- [ForgotPassword](#)
- [ConfirmForgotPassword](#)
- [ResendConfirmationCode](#)

Accepting additional user context data (AWS Management Console)

Your user pool accepts an IP address in a `UserContextData` parameter after you activate the **Accept additional user context data** feature. You don't need to activate this feature if:

- Your users only sign in with authenticated API operations like [AdminInitiateAuth](#), and you use the `ContextData` parameter.
- You only want your unauthenticated API operations to send a device fingerprint, but not an IP address, to Amazon Cognito threat protection.

Update your app client as follows in the Amazon Cognito console to add support for additional user context data.

1. Sign in to the [Amazon Cognito console](#).
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **App clients** menu.
4. Choose or create an app client. For more information, see [Configuring a user pool app client](#).
5. Choose **Edit** from the **App client information** container.
6. In the **Advanced authentication settings** for your app client, choose **Accept additional user context data**.
7. Choose **Save changes**.

To configure your app client to accept user context data in the Amazon Cognito API, set `EnablePropagateAdditionalUserContextData` to `true` in a [CreateUserPoolClient](#) or [UpdateUserPoolClient](#) request. For information about how to work with threat protection in your

web or mobile app, see [Collecting data for threat protection in applications](#). When your app calls Amazon Cognito from your server, collect user context data from the client side. The following is an example that uses the JavaScript SDK method `getData`.

```
var EncodedData =  
    AmazonCognitoAdvancedSecurityData.getData(username, userPoolId, clientId);
```

When you design your app to use adaptive authentication, we recommend that you incorporate the latest Amazon Cognito SDK into your app.. The latest version of the SDK collects device fingerprinting information like device ID, model, and time zone. For more information about Amazon Cognito SDKs, see [Install a user pool SDK](#). Amazon Cognito threat protection only saves and assigns a risk score to events that your app submits in the correct format. If Amazon Cognito returns an error response, check that your request includes a valid secret hash and that the `IPaddress` parameter is a valid IPv4 or IPv6 address.

ContextData and UserContextData resources

- AWS Amplify SDK for Android: [GetUserContextData](#)
- AWS Amplify SDK for iOS: [userContextData](#)
- JavaScript: [amazon-cognito-advanced-security-data.min.js](#)

Viewing and exporting user event history

Amazon Cognito generates a log for each authentication event by a user when you enable threat protection. By default, you can view user logs in the **Users** menu in the Amazon Cognito console or with the [AdminListUserAuthEvents](#) API operation. You can also export these events to an external system like CloudWatch Logs, Amazon S3, or Amazon Data Firehose. The export feature can make security information about user activity in your application more accessible to your own security-analysis systems.

Topics

- [Viewing user event history \(AWS Management Console\)](#)
- [Viewing user event history \(API/CLI\)](#)
- [Exporting user authentication events](#)

Viewing user event history (AWS Management Console)

To see the sign-in history for a user, you can choose the user from the **Users** menu in the Amazon Cognito console. Amazon Cognito retains user event history for two years.

Date (UTC)	Event	Result	Risk level	Risk decision	Challenge	IP	Device	Location	Event feedback
Jan 23, 2018 11:43:05 PM	Sign In	Pass	-	No Risk	Password:Success	52.94.36.11	Chrome, Windows 10	London	-
Jan 23, 2018 11:42:14 PM	Sign In	Pass	-	No Risk	Password:Success	52.94.36.11	Chrome, Windows 10	London	-
Jan 18, 2018 9:21:21 PM	Sign In	Fail	High	Account Takeover	Password:Success	67.132.130.174	Mobile, Android Mobile	Seattle	-
Jan 18, 2018 9:20:28 PM	Sign In	In Progress	High	Account Takeover	Password:Success	67.132.130.174	Mobile, Android Mobile	Seattle	-
Jan 18, 2018 9:18:18 PM	Sign In	Pass	-	No Risk	Password:Success	67.132.130.174	Mobile, Android Mobile	Seattle	Invalid

5 per page < 1 2 3 >

Each sign-in event has an event ID. The event also has corresponding context data, such as location, device details, and risk detection results.

You can also correlate the event ID with the token that Amazon Cognito issued at the time that it recorded the event. The ID and access tokens include this event ID in their payload. Amazon Cognito also correlates refresh token use to the original event ID. You can trace the original event ID back to the event ID of the sign-in event that resulted in issuing the Amazon Cognito tokens. You can trace token usage within your system to a particular authentication event. For more information, see [Understanding user pool JSON web tokens \(JWTs\)](#).

Viewing user event history (API/CLI)

You can query user event history with the Amazon Cognito API operation [AdminListUserAuthEvents](#) or with the AWS Command Line Interface (AWS CLI) with [admin-list-user-auth-events](#).

AdminListUserAuthEvents request

The following request body for AdminListUserAuthEvents returns the most recent activity log for one user.

{

```
"UserPoolId": "us-west-2_EXAMPLE",
"Username": "myexampleuser",
"MaxResults": 1
}
```

admin-list-user-auth-events request

The following request for `admin-list-user-auth-events` returns the most recent activity log for one user.

```
aws cognito-identity admin-list-user-auth-events --max-results 1 --username myexampleuser
--user-pool-id us-west-2_EXAMPLE
```

Response

Amazon Cognito returns the same JSON response body to both requests. The following is an example response for a managed login sign-in event that wasn't found to contain risk factors:

```
{
    "AuthEvents": [
        {
            "EventId": "[event ID]",
            "EventType": "SignIn",
            "CreationDate": "[Timestamp]",
            "EventResponse": "Pass",
            "EventRisk": {
                "RiskDecision": "NoRisk",
                "CompromisedCredentialsDetected": false
            },
            "ChallengeResponses": [
                {
                    "ChallengeName": "Password",
                    "ChallengeResponse": "Success"
                }
            ],
            "EventContextData": {
                "IpAddress": "192.168.2.1",
                "DeviceName": "Chrome 125, Windows 10",
                "Timezone": "-07:00",
                "City": "Bellevue",
                "Country": "United States"
            }
        }
    ]
}
```

```
        },
    ],
    "NextToken": "[event ID]#[Timestamp]"
}
```

Exporting user authentication events

Configure your user pool to export user events from threat protection to an external system. The supported external systems—Amazon S3, CloudWatch Logs, and Amazon Data Firehose—might add costs to your AWS bill for data that you send or retrieve. For more information, see [Exporting threat protection user activity logs](#).

AWS Management Console

1. Sign in to the [Amazon Cognito console](#).
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Log streaming** menu. Select **Edit**.
5. Under **Logging status**, select the checkbox next to **Activate user activity log export**.
6. Under **Logging destination**, choose the AWS service that you want to handle your logs: **CloudWatch log group**, **Amazon Data Firehose stream**, or **S3 bucket**.
7. Your selection will populate the resource selector with the corresponding resource type. Select a log group, stream, or bucket from the list. You can also select the **Create** button to navigate to the AWS Management Console for the selected service and create a new resource.
8. Select **Save changes**.

API

Choose one type of destination for your user activity logs.

The following is an example `SetLogDeliveryConfiguration` request body that sets a Firehose stream as the log destination.

```
{
    "LogConfigurations": [
```

```
{  
    "EventSource": "userAuthEvents",  
    "FirehoseConfiguration": {  
        "StreamArn": "arn:aws:firehose:us-west-2:123456789012:deliverystream/  
example-user-pool-activity-exported"  
    },  
    "LogLevel": "INFO"  
}  
],  
"UserPoolId": "us-west-2_EXAMPLE"  
}
```

The following is an example SetLogDeliveryConfiguration request body that sets a Amazon S3 bucket as the log destination.

```
{  
    "LogConfigurations": [  
        {  
            "EventSource": "userAuthEvents",  
            "S3Configuration": {  
                "BucketArn": "arn:aws:s3:::amzn-s3-demo-logging-bucket"  
            },  
            "LogLevel": "INFO"  
        }  
    ],  
    "UserPoolId": "us-west-2_EXAMPLE"  
}
```

The following is an example SetLogDeliveryConfiguration request body that sets a CloudWatch log group as the log destination.

```
{  
    "LogConfigurations": [  
        {  
            "EventSource": "userAuthEvents",  
            "CloudWatchLogsConfiguration": {  
                "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:DOC-  
EXAMPLE-LOG-GROUP"  
            },  
            "LogLevel": "INFO"  
        }  
    ],  
}
```

```
"UserPoolId": "us-west-2_EXAMPLE"  
}
```

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.

Note

Your event feedback influences the risk level that Amazon Cognito assigns to subsequent user sessions with the same characteristics.

In the Amazon Cognito console, choose a user from the **Users** menu and select **Provide event feedback**. You can review the event details and **Set as valid** or **Set as invalid**.

The console lists the sign-in history in user details in the **Users** menu. If you select an entry, you can mark the event as valid or not valid. You can also provide feedback through the user pool API operation [AdminUpdateAuthEventFeedback](#), and through the AWS CLI command [admin-update-auth-event-feedback](#).

When you select **Set as valid** in the Amazon Cognito console or provide a `FeedbackValue` value of `valid` in the API, you tell Amazon Cognito that you trust a user session where Amazon Cognito has evaluated some level of risk. When you select **Set as invalid** in the Amazon Cognito console or provide a `FeedbackValue` value of `invalid` in the API, you tell Amazon Cognito that you don't trust a user session, or you don't believe that Amazon Cognito evaluated a high-enough risk level.

Sending notification messages

With threat protection, Amazon Cognito can notify your users of risky sign-in attempts. Amazon Cognito can also prompt users to select links to indicate if the sign-in was valid or not valid. Amazon Cognito uses this feedback to improve the risk detection accuracy for your user pool.

In the **Automatic risk response** section choose **Notify Users** for low, medium, or high-risk cases.

Automatic risk response <small>Info</small>					
Risk level	Allow sign-in	Optional MFA	Require MFA	Block sign-in	Notify user
Low risk	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>
Medium risk	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>
High risk	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="checkbox"/>

Amazon Cognito sends email notifications to your users regardless of whether they have verified their email address.

You can customize notification email messages, and provide both plaintext and HTML versions of these messages. To customize your email notifications, open **Email templates from Adaptive authentication messages** in your threat protection configuration. To learn more about email templates, see [Message templates](#).

Viewing threat protection metrics

The metrics that your user pool publishes have statistical information about the effect that your threat protection settings have on user authentication activity. You might want to know how many users are attempting to sign in with compromised credentials. You can also find out what percentage of sign-in activity was evaluated to have some level of risk. Amazon Cognito publishes metrics for threat protection features to your account in Amazon CloudWatch. Amazon Cognito groups the threat protection metrics together by risk level and also by request level.

To add context to your risk analysis, you can [view information about individual user sign-in attempts](#), either in your user pool or in an exported data source.

To view metrics in 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.

Metric	Description	Metric Dimensions
Risk	Requests that Amazon Cognito marked as risky.	<p>RiskLevel: high, medium, or low.</p> <p>Operation: The type of operation, such as <code>PasswordChange</code>, <code>SignIn</code>, or <code>SignUp</code>.</p> <p>UserPoolId: The identifier of the user pool.</p>
NoRisk	Requests where Amazon Cognito did not identify any risk.	<p>Operation: The type of operation, such as <code>PasswordChange</code>, <code>SignIn</code>, or <code>SignUp</code>.</p> <p>UserPoolId: The identifier of the user pool.</p>

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 identifies as risky. **By Request Classification** reflects metrics aggregated by request level.

Aggregated Metrics Group	Description
By Risk Classification	Requests that Amazon Cognito identifies as risky.
By Request Classification	Metrics aggregated by request.

Collecting data for threat protection in applications

Amazon Cognito [adaptive authentication](#) evaluates risk levels for attempted account takeover from contextual details of users' sign-in attempts. Your application must add *context data* to API requests so that Amazon Cognito threat protection can more accurately evaluate risk. Context

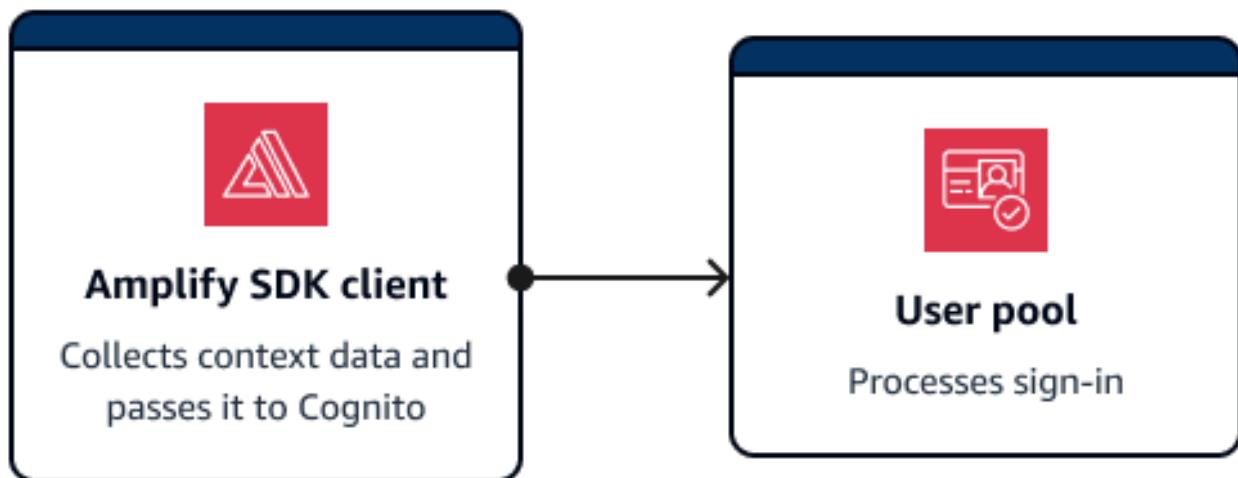
data is information like IP address, browser agent, device information, and request headers that provides contextual information about how a user connected to your user pool.

The central responsibility of an application that submits this context to Amazon Cognito is an `EncodedData` parameter in authentication requests to user pools. To add this data to your requests, you can implement Amazon Cognito with an SDK that automatically generates this information for you, or you can implement a module for JavaScript, iOS, or Android that collects this data. *Client-only* applications that make direct requests to Amazon Cognito must implement AWS Amplify SDKs. *Client-server* applications that have an intermediate server or API component must implement a separate SDK module.

In the following scenarios, your authentication front end manages user context data collection without any additional configuration:

- Managed login automatically collects and submits context data to threat protection.
- All AWS Amplify libraries have context-data collection built into their authentication methods.

Submitting user context data in client-only applications with Amplify



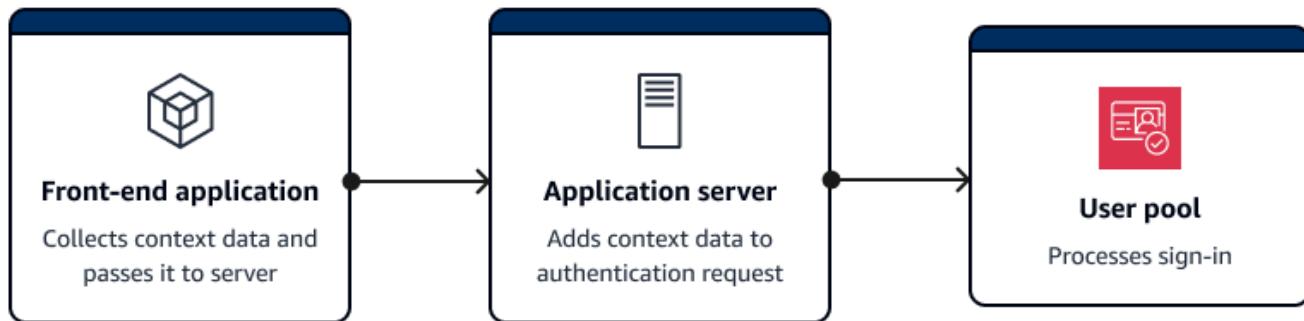
Amplify SDKs support mobile clients that authenticate with Amazon Cognito directly. Clients of this kind make direct API requests to Amazon Cognito public API operations. Amplify clients automatically collect context data for threat protection by default.

Amplify applications with JavaScript are an exception. They require the addition of a [JavaScript module](#) that collects user context data.

Typically, an application in this configuration uses unauthenticated API operations like [InitiateAuth](#) and [RespondToAuthChallenge](#). The [UserContextData](#) object helps evaluate risks more accurately for these operations. The Amplify SDKs add device and session information to an `EncodedData` parameter of `UserContextData`.

Collecting context data in client-server applications

Some applications have a front-end tier that collects user authentication data and an application back-end tier that submits authentication requests to Amazon Cognito. This is a common architecture in webservers and applications backed by microservices. In these applications, you must import a public context-data collection library.



Typically, an application server in this configuration uses authenticated API operations like [AdminInitiateAuth](#) and [AdminRespondToAuthChallenge](#). The [ContextData](#) object helps Amazon Cognito evaluate risks more accurately for these operations. The contents of `ContextData` are the encoded data that your front end passed to your server, and additional details from the user's HTTP request to your server. These additional context details, like the HTTP headers and IP address, provide your application server with the characteristics of the user's environment.

Your application server might also do sign-in with unauthenticated API operations like [InitiateAuth](#) and [RespondToAuthChallenge](#). The [UserContextData](#) object informs threat protection risk analysis in these operations. The operations in the available public context data collection libraries add security information to the `EncodedData` parameter in authentication requests. Additionally, configure your user pool to accept additional context data and add the user's source IP to the `IpAddress` parameter of `UserContextData`.

To add context data to client-server applications

1. In your front-end application, collect encoded context data from the client with an [iOS, Android, or JavaScript module](#).
2. Pass the encoded data and the details of the authentication request to your application server.
3. In your application server, extract the user's IP address, relevant HTTP headers, requested server name, and requested path from the HTTP request. Populate these values to the [ContextData](#) parameter of your API request to Amazon Cognito.
4. Populate the EncodedData parameter of ContextData in your API request with the encoded device data that your SDK module collected. Add this context data to the authentication request.

Context data libraries for client-server applications

JavaScript

The `amazon-cognito-advanced-security-data.min.js` module collects EncodedData that you can pass to your application server.

Add the `amazon-cognito-advanced-security-data.min.js` module to your JavaScript configuration. Replace `<region>` with an AWS Region from the following list: `us-east-1`, `us-east-2`, `us-west-2`, `eu-west-1`, `eu-west-2`, or `eu-central-1`.

```
<script src="https://amazon-cognito-assets.<region>.amazoncognito.com/amazon-cognito-advanced-security-data.min.js"></script>
```

To generate an `encodedContextData` object that you can use in the `EncodedData` parameter, add the following to your JavaScript application source:

```
var encodedContextData = AmazonCognitoAdvancedSecurityData.getData(_username,  
_userpoolId, _userPoolClientId);
```

iOS/Swift

To generate context data, iOS applications can integrate the [Mobile SDK for iOS](#) module [AWSCognitoIdentityProviderASF](#).

To collect encoded context data for threat protection, add the following snippet to your application:

```
import AWSCognitoIdentityProviderASF

let deviceId = getDeviceId()
let encodedContextData = AWSCognitoIdentityProviderASF.userContextData(
    userPoolId,
    username: username,
    deviceId: deviceId,
    userPoolClientId: userPoolClientId)

< /**
 * Reuse DeviceId from keychain or generate one for the first time.
 */
func getDeviceId() -> String {
    let deviceIdKey = getKeyChainKey(namespace: userPoolId, key:
"AWSCognitoAuthAsfDeviceId")

    if let existingDeviceId = self.keychain.string(forKey: deviceIdKey) {
        return existingDeviceId
    }

    let newDeviceId = UUID().uuidString
    self.keychain.setString(newDeviceId, forKey: deviceIdKey)
    return newDeviceId
}

< /**
 * Get a namespaced keychain key given a namespace and key
 */
func getKeyChainKey(namespace: String, key: String) -> String {
    return "\(namespace).\\"(key)"
}
```

Android

To generate context data, Android applications can integrate the [Mobile SDK for Android](#) module [aws-android-sdk-cognitoidentityprovider-asf](#).

To collect encoded context data for threat protection, add the following snippet to your application:

```
UserContextDataProvider provider = UserContextDataProvider.getInstance();
// context here is android application context.
```

```
String encodedContextData = provider.getEncodedContextData(context, username,
    userPoolId, userPoolClientId);
```

Associating an AWS WAF web ACL with a user pool

AWS WAF is a web application firewall. With an AWS WAF web access control list (web ACL), you can protect your user pool from unwanted requests to your managed login and Amazon Cognito API service endpoints. A web ACL gives you fine-grained control over all of the HTTPS web requests that your user pool responds to. For more information about AWS WAF web ACLs, see [Managing and using a web access control list \(web ACL\)](#) in the *AWS WAF Developer Guide*.

When you have an AWS WAF web ACL associated with a user pool, Amazon Cognito forwards selected non-confidential headers and contents of requests from your users to AWS WAF. AWS WAF inspects the contents of the request, compares it to the rules that you specified in your web ACL, and returns a response to Amazon Cognito.

Things to know about AWS WAF web ACLs and Amazon Cognito

- Requests blocked by AWS WAF do not count towards the request rate quota for any request type. The AWS WAF handler is called before the API-level throttling handlers.
- When you create a web ACL, a small amount of time passes before the web ACL has fully propagated and is available to Amazon Cognito. The propagation time can be from a few seconds to a number of minutes. AWS WAF returns a [WAFUnavailableEntityException](#) when you attempt to associate a web ACL before it has fully propagated.
- You can associate one web ACL with a user pool.
- Your request might result in a payload that is larger than the limits of what AWS WAF can inspect. See [Oversize request component handling](#) in the *AWS WAF Developer Guide* to learn how to configure how AWS WAF handles oversize requests from Amazon Cognito.
- You can't associate a web ACL that uses AWS WAF [Fraud Control account takeover prevention \(ATP\)](#) with an Amazon Cognito user pool. You implement the ATP feature when you add the AWS-AWSManagedRulesATPRuleSet managed rule group. Before you associate it with a user pool, ensure that your web ACL doesn't use this managed rule group.
- When you have an AWS WAF web ACL associated with a user pool, and a rule in your web ACL presents a CAPTCHA, this can cause an unrecoverable error in managed login TOTP registration. To create a rule that has a CAPTCHA action and doesn't affect managed login TOTP, see [Configuring your AWS WAF web ACL for managed login TOTP MFA](#).

AWS WAF inspects requests to the following endpoints.

Managed login

Requests to all endpoints in the [User pool endpoints and managed login reference](#).

Public API operations

Requests from your app to the Amazon Cognito API that don't use AWS credentials to authorize.

This includes API operations like [InitiateAuth](#), [RespondToAuthChallenge](#), and [GetUser](#).

The API operations that are in scope of AWS WAF don't require authentication with AWS credentials. They are unauthenticated, or authorized with a session string or access token. For more information, see [Amazon Cognito user pools authenticated and unauthenticated API operations](#).

You can configure the rules in your web ACL with rule actions that **Count**, **Allow**, **Block**, or present a **CAPTCHA** in response to a request that matches a rule. For more information, see [AWS WAF rules](#) in the *AWS WAF Developer Guide*. Depending on the rule action, you can customize the response that Amazon Cognito returns to your users.

Important

Your options to customize the error response depends on the way you make an API request.

- You can customize the error code and response body of managed login requests. You can only present a CAPTCHA for your user to solve in managed login.
- For requests that you make with the Amazon Cognito [user pools API](#), you can customize the response body of a request that receives a **Block** response. You can also specify a custom error code in the range 400–499.
- The AWS Command Line Interface (AWS CLI) and the AWS SDKs return a `ForbiddenException` error to requests that produce a **Block** or **CAPTCHA** response.

Associating a web ACL with your user pool

To work with a web ACL in your user pool, your AWS Identity and Access Management (IAM) principal must have the following Amazon Cognito and AWS WAF permissions. For information about AWS WAF permissions, see [AWS WAF API permissions](#) in the *AWS WAF Developer Guide*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowWebACLUserPool",  
            "Effect": "Allow",  
            "Action": [  
                "cognito-idp>ListResourcesForWebACL",  
                "cognito-idp:GetWebACLForResource",  
                "cognito-idp:AssociateWebACL"  
            ],  
            "Resource": [  
                "arn:aws:cognito-idp:*:123456789012:userpool/*"  
            ]  
        },  
        {  
            "Sid": "AllowWebACLUserPoolWAFv2",  
            "Effect": "Allow",  
            "Action": [  
                "wafv2>ListResourcesForWebACL",  
                "wafv2:AssociateWebACL",  
                "wafv2:DisassociateWebACL",  
                "wafv2:GetWebACLForResource"  
            ],  
            "Resource": "arn:aws:wafv2:*:123456789012:*/webacl/*/*"  
        },  
        {  
            "Sid": "DisassociateWebACL1",  
            "Effect": "Allow",  
            "Action": "wafv2:DisassociateWebACL",  
            "Resource": "*"  
        },  
        {  
            "Sid": "DisassociateWebACL2",  
            "Effect": "Allow",  
            "Action": [  
                "cognito-idp:DisassociateWebACL"  
            ],  
            "Resource": [  
                "arn:aws:cognito-idp:*:123456789012:userpool/*"  
            ]  
        }  
    ]  
}
```

{}

Though you must grant IAM permissions, the listed actions are permission-only and don't correspond to an [API operation](#).

To activate AWS WAF for your user pool and associate a web ACL

1. Sign in to the [Amazon Cognito console](#).
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **AWS WAF** tab in the **Security** section.
4. Choose **Edit**.
5. Select **Use AWS WAF with your user pool**.

The screenshot shows the AWS Cognito User Pool configuration page. Under the 'AWS WAF' tab, there is a section titled 'AWS WAF' with the sub-section 'AWS WAF'. A checkbox labeled 'Use AWS WAF with your user pool - Recommended' is checked. Below it, a note says 'Activate support for AWS WAF web ACLs in this user pool. AWS WAF can add cost to your bill.' with a link 'Learn more about AWS WAF pricing'. Under 'AWS WAF Web ACL', there is a dropdown menu containing 'demo-webacl', a 'View Web ACL' button, and a 'Create Web ACL in AWS WAF' button.

6. Choose an **AWS WAF Web ACL** that you already created, or choose **Create web ACL in AWS WAF** to create one in a new AWS WAF session in the AWS Management Console.
7. Choose **Save changes**.

To programmatically associate a web ACL with your user pool in the AWS Command Line Interface or an SDK, use [AssociateWebACL](#) from the AWS WAF API. Amazon Cognito doesn't have a separate API operation that associates a web ACL.

Testing and logging AWS WAF web ACLs

When you set a rule action to **Count** in your web ACL, AWS WAF adds the request to a count of requests that match the rule. To test a web ACL with your user pool, set rule actions to **Count** and

consider the volume of requests that match each rule. For example, if a rule that you want to set to a **Block** action matches a large number of requests that you determine to be normal user traffic, you might need to reconfigure your rule. For more information, see [Testing and tuning your AWS WAF protections](#) in the *AWS WAF Developer Guide*.

You can also configure AWS WAF to log request headers to an Amazon CloudWatch Logs log group, an Amazon Simple Storage Service (Amazon S3) bucket, or an Amazon Data Firehose. You can identify the Amazon Cognito requests that you make with the user pools API by the x-amzn-cognito-client-id and x-amzn-cognito-operation-name. Managed login requests only include the x-amzn-cognito-client-id header. For more information, see [Logging web ACL traffic](#) in the *AWS WAF Developer Guide*.

AWS WAF web ACLs are available in all user pool [feature plans](#). The security features of AWS WAF complement Amazon Cognito threat protection. You can activate both features in a user pool. AWS WAF bills separately for the inspection of user pool requests. For more information, see [AWS WAF Pricing](#).

Logging AWS WAF request data is subject to additional billing by the service where you target your logs. For more information, see [Pricing for logging web ACL traffic information](#) in the *AWS WAF Developer Guide*.

User pool case sensitivity

Amazon Cognito user pools that you create in the AWS Management Console are case insensitive by default. When a user pool is case insensitive, *user@example.com* and *User@example.com* refer to the same user. When usernames in a user pool are case insensitive, the `preferred_username` and `email` attributes also are case insensitive.

To account for user pool case sensitivity settings, identify users in your app code based on an alternative user attribute. Because the case of a username, preferred username, or email address attribute can vary in different user profiles, refer instead to the `sub` attribute. You can also create an immutable custom attribute in your user pool, and assign your own unique identifier value to the attribute in each new user profile. When you first create a user, you can write a value to the immutable custom attribute that you created.

Note

Regardless of the case sensitivity settings of your user pool, Amazon Cognito requires that a federated user from a SAML or OIDC identity provider (IdP) pass a unique and

case-sensitive NameId or sub claim. For more information about unique identifier case sensitivity and SAML IdPs, see [Using SP-initiated SAML sign-in](#).

Creating a case-sensitive user pool

If you create resources with the AWS Command Line Interface (AWS CLI) and API operations such as [CreateUserPool](#), you must set the Boolean CaseSensitive parameter to false. This setting creates a case-insensitive user pool. If you do not specify a value, CaseSensitive defaults to true. This default is the opposite of the default behavior for user pools that you create in the AWS Management Console. Before February 12, 2020, user pools defaulted to case sensitive regardless of platform.

You can use the **Sign-in** menu of the AWS Management Console or the [DescribeUserPool](#) API operation to review the case sensitivity settings for each user pool in your account.

Migrating to a new user pool

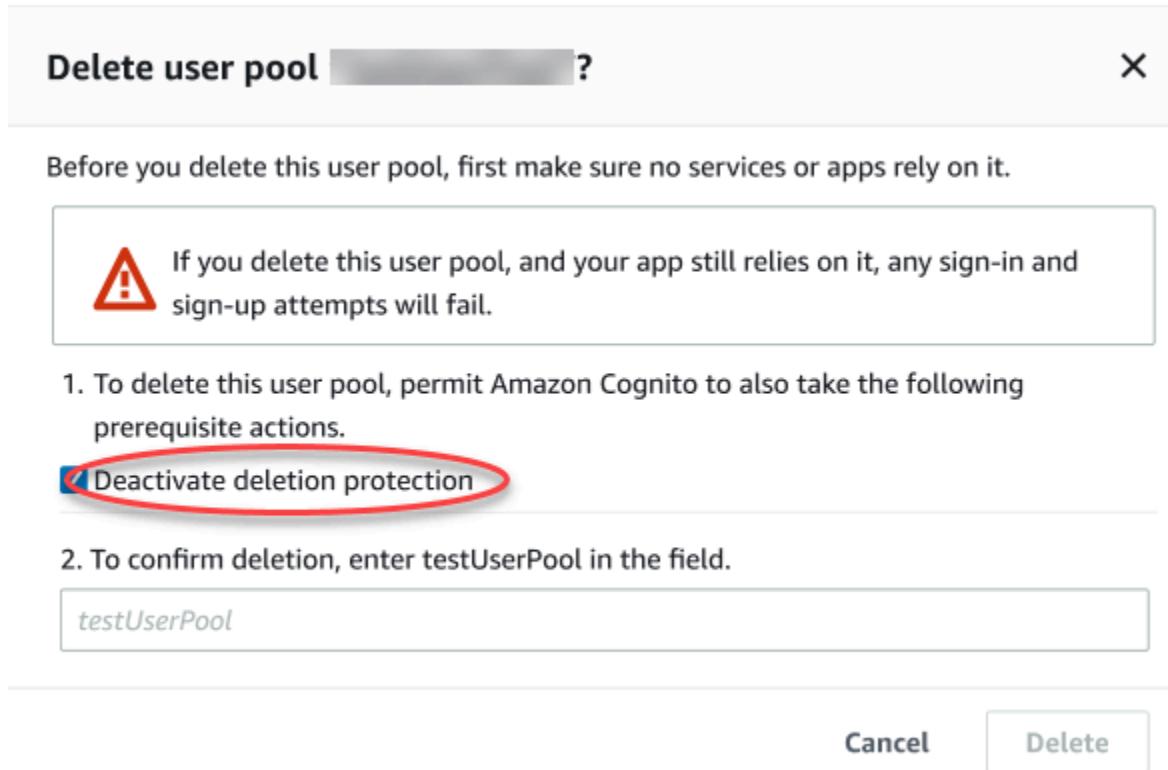
Because of potential conflicts between user profiles, you can't change an Amazon Cognito user pool from case-sensitive to case-insensitive. Instead, migrate your users to a new user pool. You must build migration code to resolve case-related conflicts. This code must either return a unique new user or reject the sign-in attempt when it detects a conflict. In a new case-insensitive user pool, assign a [Migrate user Lambda trigger](#). The AWS Lambda function can create users in the new case-insensitive user pool. When the user fails sign-in with the case-insensitive user pool, the Lambda function finds and duplicates the user from the case-sensitive user pool. You can also activate a migrate user Lambda trigger on [ForgotPassword](#) events. Amazon Cognito passes user information and event metadata from the sign-in or password-recovery action to your Lambda function. You can use event data to manage conflicts between usernames and email addresses when your function creates the new user in your case-insensitive user pool. These conflicts are between usernames and email addresses that would be unique in a case-insensitive user pool, but identical in a case-sensitive user pool.

For more information about how to use a migrate user Lambda trigger between Amazon Cognito user pools, see [Migrating Users to Amazon Cognito user pools](#) in the AWS blog.

User pool deletion protection

To make it so that your administrators don't accidentally delete your user pool, activate deletion protection. With deletion protection active, you must confirm that you want to delete your user

pool before you delete it. When you delete a user pool in the AWS Management Console, you can deactivate deletion protection at the same time. When you accept the prompt to deactivate deletion protection and confirm your intention to delete, as shown in the following image, Amazon Cognito deletes your user pool.



When you want to delete a user pool with an Amazon Cognito API request, you must first change `DeletionProtection` to `Inactive` in an [UpdateUserPool](#) request. If you don't deactivate deletion protection, Amazon Cognito returns an `InvalidParameterException` error. After you deactivate deletion protection, you can delete the user pool in a [DeleteUserPool](#) request.

Amazon Cognito activates **Deletion protection** by default when you create a new user pool in the AWS Management Console. When you create a user pool with the `CreateUserPool` API, deletion protection is inactive by default. To use this feature in user pools that you create with the AWS CLI or an AWS SDK, set the `DeletionProtection` parameter to `True`.

You can activate or deactivate deletion protection status in the **Deletion protection** container in the **Settings** menu in the Amazon Cognito console.

To configure deletion protection

1. Go to the [Amazon Cognito console](#). You might be prompted for your AWS credentials.

2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Settings** menu and navigate to the **Deletion Protection** tab. Select **Activate** or **Deactivate**.
5. Confirm your choice in the next dialogue.

Managing user existence error responses

Amazon Cognito supports customizing error responses returned by user pools. Custom error responses are available for user creation and authentication, password recovery, and confirmation operations.

Use the `PreventUserExistenceErrors` setting of a user pool app client to enable or disable user existence related errors. When you create a new app client with the Amazon Cognito user pools API, `PreventUserExistenceErrors` is LEGACY, or disabled, by default. In the Amazon Cognito console, the option **Prevent user existence errors**—a setting of ENABLED for `PreventUserExistenceErrors`—is selected by default. To update your `PreventUserExistenceErrors` configuration, do one of the following:

- Change the value of `PreventUserExistenceErrors` between ENABLED and LEGACY in an [UpdateUserPoolClient](#) API request.
- Edit your app client in the Amazon Cognito console and change the state of **Prevent user existence errors** between selected (ENABLED) and deselected (LEGACY).

When this property has a value of LEGACY, your app client returns a `UserNotFoundException` error response when a user attempts to sign in with a username that doesn't exist in your user pool.

When this property has a value of ENABLED, your app client doesn't disclose the nonexistence of a user account in your user pool with a `UserNotFoundException` error. A `PreventUserExistenceErrors` configuration of ENABLED has the following effects:

- Amazon Cognito responds with nonspecific information to API requests where its response might otherwise disclose that a valid user exists.
- Amazon Cognito sign-in and forgot-password 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, instead of a partial representation of a user's contact information.

The following information details the behaviors of user pool operations when `PreventUserExistenceErrors` is set to `ENABLED`.

Authentication and user creation operations

You can configure error responses in both *Username-password* and *Secure Remote Password (SRP)* authentication. You can also customize the errors that you return with custom authentication. The following APIs perform these authentication operations:

- `AdminInitiateAuth`
- `AdminRespondToAuthChallenge`
- `InitiateAuth`
- `RespondToAuthChallenge`

The following list demonstrates how you can customize error responses in user authentication operations.

Username and password authentication

To sign a user in with `ADMIN_USER_PASSWORD_AUTH` and `USER_PASSWORD_AUTH`, include the username and password in an `AdminInitiateAuth` or `InitiateAuth` API request. Amazon Cognito returns a generic `NotAuthorizedException` error when either the username or password is incorrect.

Secure Remote Password (SRP) based authentication

To sign a user in with `USER_SRP_AUTH`, include a username and an `SRP_A` parameter in an `AdminInitiateAuth` or `InitiateAuth` API request. In response, Amazon Cognito returns `SRP_B` and salt for the user. 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 an internal user ID in [UUID](#) format for the same username and user pool combination. When you send a `RespondToAuthChallenge` API request with proof of password, Amazon Cognito returns a generic `NotAuthorizedException` error when either username or password is incorrect.

Note

You can simulate a generic response with username and password authentication if you are using verification-based alias attributes, and the immutable username isn't formatted as a [UUID](#).

Custom Authentication Challenge Lambda trigger

If you use the [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.

The following list demonstrates how you can customize error responses in user creation operations.

SignUp

The SignUp operation always returns UsernameExistsException when a username is already taken. If you don't want Amazon Cognito to return a UsernameExistsException error for email addresses and phone numbers when you sign up users in your app, use verification-based alias attributes. For more information about aliases, see [Customizing sign-in attributes](#).

For an example of how Amazon Cognito can prevent the use of SignUp API requests to discover users in your user pool, see [Preventing UsernameExistsException errors for email addresses and phone numbers on sign-up](#).

Imported users

If PreventUserExistenceErrors is enabled, during authentication of imported users a generic NotAuthorizedException error is returned indicating either the username 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](#).

Preventing `UsernameExistsException` errors for email addresses and phone numbers on sign-up

The following example demonstrates how, when you configure alias attributes in your user pool, you can keep duplicate email addresses and phone numbers from generating `UsernameExistsException` errors in response to SignUp API requests. You must have created your user pool with email address or phone number as an alias attribute. For more information, see the *Customizing sign-in attributes* section of [User pool attributes](#).

1. Jie signs up for a new username, and also provides the email address `jie@example.com`. Amazon Cognito sends a code to their email address.

Example AWS CLI command

```
aws cognito-idp sign-up --client-id 1234567890abcdef0 --username jie --password
PASSWORD --user-attributes Name="email",Value="jie@example.com"
```

Example response

```
{
    "UserConfirmed": false,
    "UserSub": "<subId>",
    "CodeDeliveryDetails": {
        "AttributeName": "email",
        "Destination": "j*****@e*****",
        "DeliveryMedium": "EMAIL"
    }
}
```

2. Jie provides the code sent to them to confirm their ownership of the email address. This completes their registration as a user.

Example AWS CLI command

```
aws cognito-idp confirm-sign-up --client-id 1234567890abcdef0 --username=jie --confirmation-code xxxxxx
```

3. Shirley registers a new user account and provides the email address `jie@example.com`. Amazon Cognito doesn't return a `UsernameExistsException` error, and sends a confirmation code to Jie's email address.

Example AWS CLI command

```
aws cognito-idp sign-up --client-id 1234567890abcdef0 --username shirley --password PASSWORD --user-attributes Name="email",Value="jie@example.com"
```

Example response

```
{  
    "UserConfirmed": false,  
    "UserSub": "<new subId>",  
    "CodeDeliveryDetails": {  
        "AttributeName": "email",  
        "Destination": "j*****@e*****",  
        "DeliveryMedium": "EMAIL"  
    }  
}
```

4. In a different scenario, Shirley has ownership of `jie@example.com`. Shirley retrieves the code that Amazon Cognito sent to Jie's email address and attempts to confirm the account.

Example AWS CLI command

```
aws cognito-idp confirm-sign-up --client-id 1234567890abcdef0 --username=shirley --confirmation-code xxxxxx
```

Example response

An error occurred (`AliasExistsException`) when calling the `ConfirmSignUp` operation: An account with the email already exists.

Amazon Cognito doesn't return an error to Shirley's aws cognito-idp sign-up request, despite jie@example.com being assigned to an existing user. Shirley must demonstrate ownership of the email address before Amazon Cognito returns an error response. In a user pool with alias attributes, this behavior prevents use of the public SignUp API to check whether a user exists with a given email address or phone number.

This behavior is different from the response that Amazon Cognito returns to SignUp request with an existing username, as shown in the following example. While Shirley learns from this response that a user already exists with the username jie, they don't learn about any email addresses or phone numbers associated with the user.

Example CLI command

```
aws cognito-idp sign-up --client-id 1example23456789 --username jie --password PASSWORD  
--user-attributes Name="email",Value="shirley@example.com"
```

Example response

```
An error occurred (UsernameExistsException) when calling the SignUp operation: User  
already exists
```

Password reset operations

Amazon Cognito returns the following responses to user password reset operations when you prevent user existence errors.

ForgotPassword

When a user isn't found, is deactivated, or doesn't have a verified delivery 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 username 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.

Confirmation operations

Amazon Cognito returns the following responses to user confirmation and verification operations when you prevent user existence errors.

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`.

User pool endpoints and managed login reference

Amazon Cognito has two models of user pool authentication: with the user pools API and with the OAuth 2.0 authorization server. Use the API when you want to retrieve OpenID Connect (OIDC) tokens with an AWS SDK in your application back end. Use the authorization server when you want to implement your user pool as an OIDC provider. The authorization server adds features like [federated sign-in](#), [API and M2M authorization with access token scopes](#), and [managed login](#). You can use the API and OIDC models each on their own or together, configured at the user pool level or at the [app client](#) level. This section is a reference for the implementation of the OIDC model. For more information about the two authentication models, see [Understanding API, OIDC, and managed login pages authentication](#).

Amazon Cognito activates the public webpages listed here when you assign a domain to your user pool. Your domain serves as a central access point for all of your app clients. They include managed login, where your users can sign up and sign in ([Login endpoint](#)), and sign out ([Logout endpoint](#)). For more information about these resources, see [User pool managed login](#).

These pages also include the public web resources that allow your user pool to communicate with third-party SAML, OpenID Connect (OIDC) and OAuth 2.0 identity providers (IdPs). To sign in a user with a federated identity provider, your users must initiate a request to the interactive managed login [Login endpoint](#) or the OIDC [Authorize endpoint](#). The Authorize endpoint redirects your users either to your managed login pages or your IdP sign-in page.

Your app can also sign in *local users* with the [Amazon Cognito user pools API](#). A local user exists exclusively in your user pool directory without federation through an external IdP.

In addition to managed login, Amazon Cognito integrates with SDKs for Android, iOS, JavaScript, and more. The SDKs provide tools to perform user pool API operations with Amazon Cognito API service endpoints. For more information about service endpoints, see [Amazon Cognito Identity endpoints and quotas](#).

Warning

Don't pin the end-entity or intermediate Transport Layer Security (TLS) certificates for Amazon Cognito domains. AWS manages all certificates for all of your user pool endpoints and prefix domains. The certificate authorities (CAs) in the chain of trust that supports Amazon Cognito certificates dynamically rotate and renew. When you pin your app to an intermediate or leaf certificate, your app can fail without notice when AWS rotates certificates.

Instead, pin your application to all available [Amazon root certificates](#). For more information, see best practices and recommendations at [Certificate pinning](#) in the [AWS Certificate Manager User Guide](#).

Topics

- [User-interactive managed login and managed login endpoints](#)
- [Identity provider and relying party endpoints](#)
- [OAuth 2.0 grants](#)
- [Using PKCE in authorization code grants](#)
- [Managed login and federation error responses](#)

User-interactive managed login and managed login endpoints

Amazon Cognito activates the managed login endpoints in this section when you add a domain to your user pool. They are webpages where your users can complete the core authentication operations of a user pool. They include pages for password management, multi-factor authentication (MFA), and attribute verification.

The webpages that make up managed login are a front-end web application for interactive user sessions with your customers. Your app must invoke managed login in your users' browsers.

Amazon Cognito doesn't support programmatic access to the webpages in this chapter. Those federation endpoints in the [Identity provider and relying party endpoints](#) that return a JSON response can be queried directly in your app code. The [Authorize endpoint](#) redirects either to managed login or to an IdP sign-in page and also must be opened in users' browsers.

All user pool endpoints accept traffic from IPv4 and IPv6 source IP addresses.

The topics in this guide describe frequently-used managed login endpoints in detail. Amazon Cognito makes the webpages that follow available when you assign a domain to your user pool.

Managed login endpoints

Endpoint URL	Description	How it's accessed
<code>https://Your user pool domain/login</code>	Signs in user pool local and federated users.	Redirect from endpoints like Authorize endpoint , /logout, and /confirmForgotPassword . See Login endpoint .
<code>https://Your user pool domain/logout</code>	Signs out user pool users.	Direct link. See Logout endpoint .
<code>https://Your user pool domain/confirmUser</code>	Confirms users who have selected an email link to verify their user account.	User selected link in an email message.
<code>https://Your user pool domain/signup</code>	Signs up a new user. The /login page directs your user to /signup when they select Sign up .	Direct link with same parameters as /oauth2/authorize .
<code>https://Your user pool domain/confirm</code>	After your user pool sends a confirmation code to a user who signed up, prompts your user for the code.	Redirect-only from /signup.
<code>https://Your user pool domain/forgotPassword</code>	Prompts your user for their user name and sends a password-reset code. The	1. From Forgot password link at /login.

Endpoint URL	Description	How it's accessed
	/login page directs your user to /forgotPassword when they select Forgot your password? .	2. Direct link with same parameters as /oauth2/authorize .
https://Your user pool domain/confirmforgotPassword	Prompts your user for their password-reset code and a new password. The /forgotPassword page directs your user to /confirmforgotPassword when they select Reset your password .	Redirect-only from /forgotPassword .
https://Your user pool domain/resendcode	Sends a new confirmation code to a user who has signed up in your user pool.	Redirect-only from Send a new code link at /confirm.
https://Your user pool domain/passkeys/add	Registers a new passkey .	<ul style="list-style-type: none"> In the sign-up flow after confirmation in app clients that support passkey authentication. Direct link with same parameters as /oauth2/authorize .

Topics

- [The managed login sign-in endpoint: /login](#)
- [The managed login sign-out endpoint: /logout](#)

The managed login sign-in endpoint: /login

The login endpoint is an authentication server and a redirect destination from [Authorize endpoint](#). It's the entry point to managed login when you don't specify an identity provider. When you

generate a redirect to the login endpoint, it loads the login page and presents the authentication options configured for the client to the user.

Note

The login endpoint is a component of managed login. In your app, invoke federation and managed login pages that redirect to the login endpoint. Direct access by users to the login endpoint isn't a best practice.

GET /login

The /login endpoint only supports HTTPS GET for your user's initial request. Your app invokes the page in a browser like Chrome or Firefox. When you redirect to /login from the [Authorize endpoint](#), it passes along all the parameters that you provided in your initial request. The login endpoint supports all the request parameters of the authorize endpoint. You can also access the login endpoint directly. As a best practice, originate all your users' sessions at /oauth2/authorize.

Example – 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
```

Example – response

The authentication server redirects to your app with the authorization code and state. The server must return the code and state 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
```

User-initiated sign-in request

After your user loads the /login endpoint, they can enter a user name and password and choose **Sign in**. When they do this, they generate an HTTPS POST request with the same header request parameters as the GET request, and a request body with their username, password, and a device fingerprint.

The managed login sign-out endpoint: /logout

The /logout endpoint is a redirection endpoint. It signs out the user and redirects either to an authorized sign-out URL for your app client, or to the /login endpoint. The available parameters in a GET request to the /logout endpoint are tailored to Amazon Cognito managed login use cases.

To redirect your user to managed login to sign in again, add a `redirect_uri` parameter to your request. A logout request with a `redirect_uri` parameter must also include parameters for your subsequent request to the [Login endpoint](#), like `client_id`, `response_type`, and `scope`.

The logout endpoint is a front-end web application for interactive user sessions with your customers. Your app must invoke this and other managed login endpoints in your users' browsers.

To redirect your user to a page that you choose, add **Allowed sign-out URLs** to your app client. In your users' requests to the logout endpoint, add `logout_uri` and `client_id` parameters. If the value of `logout_uri` is one of the **Allowed sign-out URLs** for your app client, Amazon Cognito redirects users to that URL.

With single logout (SLO) for SAML 2.0 IdPs, Amazon Cognito first redirects your user to the SLO endpoint you defined in your IdP configuration. After your IdP redirects your user back to `saml2/logout`, Amazon Cognito responds with one more redirect to the `redirect_uri` or `logout_uri` from your request. For more information, see [Signing out SAML users with single sign-out](#).

The logout endpoint doesn't sign users out of OIDC or social identity providers (IdPs). To sign users out from their session with an external IdP, direct them to the sign-out page for that provider.

GET /logout

The /logout endpoint only supports HTTPS GET. The user pool client typically makes this request through the system browser. The browser is typically Custom Chrome Tab in Android or Safari View Control in iOS.

Request parameters

client_id

The app client ID for your app. To get an app client ID, you must register the app in the user pool. For more information, see [Application-specific settings with app clients](#).

Required.

logout_uri

Redirect your user to a custom sign-out page with a *logout_uri* parameter. Set its value to the app client **sign-out URL** where you want to redirect your user after they sign out. Use *logout_uri* only with a *client_id* parameter. For more information, see [Application-specific settings with app clients](#).

You can also use the *logout_uri* parameter to redirect your user to the sign-in page for another app client. Set the sign-in page for the other app client as an **Allowed callback URL** in your app client. In your request to the /logout endpoint, set the value of the *logout_uri* parameter to the URL-encoded sign-in page.

Amazon Cognito requires either a *logout_uri* or a *redirect_uri* parameter in your request to the /logout endpoint. A *logout_uri* parameter redirects your user to another website. If both *logout_uri* and *redirect_uri* parameters are included in your request to the /logout endpoint, Amazon Cognito will utilize the *logout_uri* parameter exclusively, overriding the *redirect_uri* parameter.

redirect_uri

Redirect your user to your sign-in page to authenticate with a *redirect_uri* parameter. Set its value to the app client **Allowed callback URL** where you want to redirect your user after they sign in again. Add *client_id*, *scope*, *state*, and *response_type* parameters that you want to pass to your /login endpoint.

Amazon Cognito requires either a *logout_uri* or a *redirect_uri* parameter in your request to the /logout endpoint. To redirect your user to your /login endpoint to reauthenticate and pass tokens to your app, add a *redirect_uri* parameter. If both *logout_uri* and *redirect_uri* parameters are included in your request to the /logout endpoint, Amazon Cognito overrides the *redirect_uri* parameter and processes the *logout_uri* parameter exclusively.

response_type

The OAuth 2.0 response that you want to receive from Amazon Cognito after your user signs in. code and token are the valid values for the *response_type* parameter.

Required if you use a *redirect_uri* parameter.

state

When your application adds a *state* parameter to a request, Amazon Cognito returns its value to your app when the /oauth2/logout endpoint redirects your user.

Add this value to your requests to guard against [CSRF](#) attacks.

You can't set the value of a *state* parameter to a URL-encoded JSON string. To pass a string that matches this format in a *state* parameter, encode the string to base64, then decode it in your application.

Strongly recommended if you use a *redirect_uri* parameter.

scope

The OAuth 2.0 scopes that you want to request from Amazon Cognito after you sign them out with a *redirect_uri* parameter. Amazon Cognito redirects your user to the /login endpoint with the *scope* parameter in your request to the /logout endpoint.

Optional if you use a *redirect_uri* parameter. If you don't include a *scope* parameter, Amazon Cognito redirects your user to the /login endpoint with a *scope* parameter. When Amazon Cognito redirects your user and automatically populates scope, the parameter includes all authorized scopes for your app client.

Example requests

Example – log out and redirect user to client

Except for *logout_uri* and *client_id*, all possible query parameters for this endpoint are passed through to the [Authorize endpoint](#). Amazon Cognito redirects user sessions to the URL in the value of *logout_uri*, ignoring all other request parameters, when requests include *logout_uri* and *client_id*. This URL must be an authorized sign-out URL for the app client.

The following is an example request for sign-out and redirect to <https://www.example.com/welcome>.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?  
client_id=1example23456789&  
logout_uri=https%3A%2F%2Fwww.example.com%2Fwelcome
```

Example – log out and prompt the user to sign in as another user

When requests omit `logout_uri` but otherwise provide the parameters that make up a well-formed request to the authorize endpoint, Amazon Cognito redirects users to managed login sign-in. The logout endpoint appends the parameters in your original request to the redirect destination. The parameter `redirect_uri` in a request to the logout endpoint is not a sign-out URL, but a sign-in URL that you want to pass through to the authorize endpoint.

The following is an example request that signs a user out, redirects to the sign-in page, and provides an authorization code to `https://www.example.com` after sign-in.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?  
response_type=code&  
client_id=1example23456789&  
redirect_uri=https%3A%2F%2Fwww.example.com&  
state=example-state-value&  
nonce=example-nonce-value&  
scope=openid+profile+aws.cognito.signin.user.admin
```

Identity provider and relying party endpoints

Federation endpoints are user pool endpoints serve a purpose for one of the authentication standards used by user pools. They include SAML ACS URLs, OIDC discovery endpoints, and service endpoints for user pool roles both as identity provider and relying party. Federation endpoints initiate authentication flows, receive proof of authentication from IdPs, and issue tokens to clients. They interact with IdPs, applications, and administrators, but not with users.

The full-page topics after this page have details about the OAuth 2.0 and OIDC provider endpoints that become available when you add a domain to your user pool. The following chart is a list of all federation endpoints.

User pool federation endpoints

Endpoint URL	Description	How it's accessed
<code>https://Your user pool domain/oauth2/authorize</code>	Redirects a user to either managed login or to sign in with their IdP.	Invoked in customer browser to begin user authentication. See Authorize endpoint .
<code>https://Your user pool domain/oauth2/token</code>	Returns tokens based on an authorization code or client credentials request.	Requested by app to retrieve tokens. See Token endpoint .
<code>https://Your user pool domain/oauth2/userInfo</code>	Returns user attributes based on OAuth 2.0 scopes and user identity in an access token.	Requested by app to retrieve user profile. See userInfo endpoint .
<code>https://Your user pool domain/oauth2/revoke</code>	Revokes a refresh token and the associated access tokens.	Requested by app to revoke a token. See Revoke endpoint .
<code>https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/openid-configuration</code>	A directory of the OIDC architecture of your user pool.	Requested by app to locate user pool issuer metadata.
<code>https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/jwks.json</code>	Public keys that you can use to validate Amazon Cognito tokens.	Requested by app to verify JWTs.
<code>https://Your user pool domain/oauth2/idpresponse</code>	Social identity providers must redirect your users to this endpoint with an authorization code. Amazon Cognito redeems the code for a token when it authenticates your federated user.	Redirected from OIDC IdP sign-in as the IdP client callback URL.

Endpoint URL	Description	How it's accessed
<code>https://Your user pool domain/saml2/idpresponse</code>	The Assertion Consumer Response (ACS) URL for integration with SAML 2.0 identity providers.	Redirected from SAML 2.0 IdP as the ACS URL, or the origination point for IdP-initiated sign-in ¹ .
<code>https://Your user pool domain/saml2/logout</code>	The Single Logout (SLO) URL for integration with SAML 2.0 identity providers.	Redirected from SAML 2.0 IdP as the single logout (SLO) URL. Accepts POST binding only.

¹ For more information about IdP-initiated SAML sign-in, see [Using IdP-initiated SAML sign-in](#).

For more information on the OpenID Connect and OAuth standards, see [OpenID Connect 1.0](#) and [OAuth 2.0](#).

Topics

- [The redirect and authorization endpoint](#)
- [The token issuer endpoint](#)
- [The user attributes endpoint](#)
- [The token revocation endpoint](#)
- [The IdP SAML assertion endpoint](#)

The redirect and authorization endpoint

The /oauth2/authorize endpoint is a redirection endpoint that supports two redirect destinations. If you include an `identity_provider` or `idp_identifier` parameter in the URL, it silently redirects your user to the sign-in page for that identity provider (IdP). Otherwise, it redirects to the [Login endpoint](#) with the same URL parameters that you included in your request.

The authorize endpoint redirects either to managed login or to an IdP sign-in page. The destination of a user session at this endpoint is a webpage that your user must interact with directly in their browser.

To use the authorize endpoint, invoke your user's browser at `/oauth2/authorize` with parameters that provide your user pool with information about the following user pool details.

- The app client that you want to sign in to.
- The callback URL that you want to end up at.
- The OAuth 2.0 scopes that you want to request in your user's access token.
- Optionally, the third-party IdP that you want to use to sign in.

You can also supply state and nonce parameters that Amazon Cognito uses to validate incoming claims.

GET /oauth2/authorize

The `/oauth2/authorize` endpoint only supports HTTPS GET. Your app typically initiates this request in your user's browser. You can only make requests to the `/oauth2/authorize` endpoint over HTTPS.

You can learn more about the definition of the authorization endpoint in the OpenID Connect (OIDC) standard at [Authorization Endpoint](#).

Request parameters

response_type

(Required) The response type. Must be code or token.

A successful request with a `response_type` of code returns an authorization code grant.

An authorization code grant is a code parameter that Amazon Cognito appends to your redirect URL. Your app can exchange the code with the [Token endpoint](#) for access, ID, and refresh tokens. As a security best practice, and to receive refresh tokens for your users, use an authorization code grant in your app.

A successful request with a `response_type` of token returns an implicit grant. An implicit grant is an ID and access token that Amazon Cognito appends to your redirect URL. An implicit grant is less secure because it exposes tokens and potential identifying information to users.

You can deactivate support for implicit grants in the configuration of your app client.

client_id

(Required) The app client ID.

The value of `client_id` must be the ID of an app client in the user pool where you make the request. Your app client must support sign-in by Amazon Cognito local users or at least one third-party IdP.

redirect_uri

(Required) The URL where the authentication server redirects the browser after Amazon Cognito authorizes the user.

A redirect uniform resource identifier (URI) must have the following attributes:

- It must be an absolute URI.
- You must have pre-registered the URI with a client.
- It can't include a fragment component.

See [OAuth 2.0 - Redirection Endpoint](#).

Amazon Cognito requires that your redirect URI use HTTPS, except for `http://localhost`, which you can set as a callback URL for testing purposes.

Amazon Cognito also supports app callback URLs such as `myapp://example`.

state

(Optional, recommended) When your app adds a `state` parameter to a request, Amazon Cognito returns its value to your app when the `/oauth2/authorize` endpoint redirects your user.

Add this value to your requests to guard against [CSRF](#) attacks.

You can't set the value of a `state` parameter to a URL-encoded JSON string. To pass a string that matches this format in a `state` parameter, encode the string to base64, then decode it in your app.

identity_provider

(Optional) Add this parameter to bypass managed login and redirect your user to a provider sign-in page. The value of the `identity_provider` parameter is the name of the identity provider (IdP) as it appears in your user pool.

- For social providers, you can use the `identity_provider` values Facebook, Google, LoginWithAmazon, and SignInWithApple.
- For Amazon Cognito user pools, use the value COGNITO.

- For SAML 2.0 and OpenID Connect (OIDC) identity providers (IdPs), use the name that you assigned to the IdP in your user pool.

idp_identifier

(Optional) Add this parameter to redirect to a provider with an alternative name for the *identity_provider* name. You can enter identifiers for your SAML 2.0 and OIDC IdPs from the **Social and external providers** menu of the Amazon Cognito console.

scope

(Optional) 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.

code_challenge_method

(Optional) The hashing protocol that you used to generate the challenge. The [PKCE RFC](#) defines two methods, `S256` and `plain`; however, Amazon Cognito authentication server supports only `S256`.

code_challenge

(Optional) The proof of key code exchange (PKCE) challenge that you generated from the `code_verifier`. For more information, see [Using PKCE in authorization code grants](#).

Required only when you specify a `code_challenge_method` parameter.

nonce

(Optional) A random value that you can add to the request. The nonce value that you provide is included in the ID token that Amazon Cognito issues. To guard against replay attacks, your app can inspect the `nonce` claim in the ID token and compare it to the one you generated. For more information about the `nonce` claim, see [ID token validation](#) in the *OpenID Connect standard*.

lang

The language that you want to display user-interactive pages in. Managed login pages can be localized, but hosted UI (classic) pages can't. For more information, see [Managed login localization](#).

login_hint

A username prompt that you want to pass to the authorization server. You can collect a username, email address or phone number from your user and allow the destination provider to pre-populate the user's sign-in name. When you submit a `login_hint` parameter and no `idp_identifier` or `identity_provider` parameters to the `oauth2/authorize` endpoint, managed login fills the username field with your hint value. You can also pass this parameter to the [Login endpoint](#) and automatically fill the username value.

When your authorization request invokes a redirect to OIDC IdPs or Google, Amazon Cognito adds a `login_hint` parameter to the request to that third-party authorizer. You can't forward login hints to SAML, Apple, Login With Amazon, or Facebook (Meta) IdPs.

Example requests with positive responses

The following examples illustrate the format of HTTP requests to the `/oauth2/authorize` endpoint.

Authorization code grant

This is an example request for an authorization code grant.

Example – GET request

The following request initiates a session to retrieve an authorization code that your user passes to your app at the `redirect_uri` destination. This session requests scopes for user attributes and for access to Amazon Cognito self-service API operations.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?  
response_type=code&  
client_id=example23456789&  
redirect_uri=https://www.example.com&  
state=abcdefg&  
scope=openid+profile+aws.cognito.signin.user.admin
```

Example – response

The Amazon Cognito authentication server redirects back to your app with the authorization code and state. The authorization code is valid for five minutes.

```
HTTP/1.1 302 Found
Location: https://www.example.com?code=a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111&state=abcdefg
```

Authorization code grant with PKCE

This is an example request for an authorization code grant with [PKCE](#).

Example – GET request

The following request adds a `code_challenge` parameter to the previous request. To complete the exchange of a code for a token, you must include the `code_verifier` parameter in your request to the `/oauth2/token` endpoint.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=aws.cognito.signin.user.admin&
code_challenge_method=S256&
code_challenge=a1b2c3d4...
```

Example – response

The authentication server redirects back to your application 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://www.example.com?code=a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111&state=abcdefg
```

Token grant without openid scope

This is an example request that generates an implicit grant and returns JWTs directly to the user's session.

Example – GET request

The following request is for an implicit grant from your authorization server. The access token from Amazon Cognito authorizes self-service API operations.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?  
response_type=token&  
client_id=1example23456789&  
redirect_uri=https://www.example.com&  
state=abcdefg&  
scope=aws.cognito.signin.user.admin
```

Example – response

The Amazon Cognito authorization server redirects back to your app with access token. Because openid scope was not requested, Amazon Cognito doesn't return an ID token. Also, Amazon Cognito doesn't return a refresh token in this flow. Amazon Cognito returns the access token and state 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

This is an example request that generates an implicit grant and returns JWTs directly to the user's session.

Example – GET request

The following request is for an implicit grant from your authorization server. The access token from Amazon Cognito authorizes access to user attributes and self-service API operations.

```
GET  
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?  
response_type=token&  
client_id=1example23456789&  
redirect_uri=https://www.example.com&  
state=abcdefg&  
scope=aws.cognito.signin.user.admin+openid+profile
```

Example – 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://www.example.com#id\_token=eyJra67890EXAMPLE&access\_token=eyJra12345EXAMPLE&token\_type=bearer&exp
```

Examples of negative responses

Amazon Cognito might deny your request. Negative requests come with an HTTP error code and a description that you can use to correct your request parameters. The following are examples of negative responses.

- If `client_id` and `redirect_uri` are valid, but the request parameters aren't formatted correctly, the authentication server redirects the error to the client's `redirect_uri` and appends an error message in a URL parameter. The following are examples of incorrect formatting.
 - The request doesn't include a `response_type` parameter.
 - The authorization request provided a `code_challenge` parameter, but not a `code_challenge_method` parameter.
 - The value of the `code_challenge_method` parameter isn't S256.

The following is the response to an example request with incorrect formatting.

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request
```

- If the client requests `code` or `token` in `response_type`, but doesn't have permission for these requests, the Amazon Cognito authorization server returns `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 scope that is unknown, malformed, or not valid, 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 the client's `redirect_uri`. Because the HTTP 500 error doesn't get sent to the client, the error doesn't display in the user's browser. The authorization server returns the following error.

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=server_error
```

- When Amazon Cognito authenticates through federation to third-party IdPs, Amazon Cognito might experience connection issues, such as the following:
 - If a connection timeout occurs while requesting token from the IdP, 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 the `jwks_uri` endpoint for ID token validation, the authentication server redirects with an 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 IdPs, the providers may return error responses. This can be due to configuration errors or other reasons, 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]
```

- When Amazon Cognito encounters a communication exception when it connects to an external IdP, the authentication server redirects with an 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`

The token issuer endpoint

The OAuth 2.0 [token endpoint](#) at `/oauth2/token` issues JSON web tokens (JWTs). These tokens are the end result of authentication with a user pool. They contain information about the user (ID token), the user's level of access (access token), and the user's entitlement to persist their signed-in session (refresh token). OpenID Connect (OIDC) relying-party libraries handle requests to and response payloads from this endpoint. Tokens provide verifiable proof of authentication, profile information, and a mechanism for access to back-end systems.

Your user pool OAuth 2.0 authorization server issues JSON web tokens (JWTs) from the token endpoint to the following types of sessions:

- Users who have completed a request for an authorization code grant. Successful redemption of a code returns ID, access, and refresh tokens.
- Machine-to-machine (M2M) sessions that have completed a client-credentials grant. Successful authorization with the client secret returns an access token.
- Users who have previously signed in and received refresh tokens. Refresh token authentication returns new ID and access tokens.

Note

Users who sign in with an authorization code grant in managed login or through federation can always refresh their tokens from the token endpoint. Users who sign in with the API operations `InitiateAuth` and `AdminInitiateAuth` can refresh their tokens with the token endpoint when [remembered devices](#) is *not* active in your user pool. If remembered devices is active, refresh tokens with the `AuthFlow` of `REFRESH_TOKEN_AUTH` in `InitiateAuth` or `AdminInitiateAuth` API requests.

The token endpoint becomes publicly available when you add a domain to your user pool. It accepts HTTP POST requests. For application security, use PKCE with your authorization code sign-in events. PKCE verifies that the user passing an authorization code is that same user who authenticated. For more information about PKCE, see [IETF RFC 7636](#).

You can learn more about the user pool app clients and their grant types, client secrets, authorized scopes, and client IDs at [Application-specific settings with app clients](#). You can learn more about M2M authorization, client credentials grants, and authorization with access token scopes at [Scopes, M2M, and APIs with resource servers](#).

To retrieve information about a user from their access token, pass it to your [userInfo endpoint](#) or to a [GetUser](#) API request.

POST /oauth2/token

The /oauth2/token endpoint only supports HTTPS POST. Your app makes requests to this endpoint directly, not through the user's browser.

The token endpoint supports `client_secret_basic` and `client_secret_post` authentication. For more information about the OpenID Connect specification, see [Client Authentication](#). For more information about the token endpoint from the OpenID Connect specification, see [Token Endpoint](#).

Request parameters in header

Authorization

If the client was issued a secret, the client can pass its `client_id` and `client_secret` in the authorization header as `client_secret_basic` HTTP authorization. You can also include the `client_id` and `client_secret` in the request body as `client_secret_post` authorization.

The authorization header string is `Basic` `Base64Encode(client_id:client_secret)`.

The following example is an authorization header for app client `djc98u3jiedmi283eu928` with client secret `abcdef01234567890`, using the Base64-encoded version of the string `djc98u3jiedmi283eu928:abcdef01234567890`:

```
Authorization: Basic ZGpj0Th1M2ppZWRtaTI4M2V10TI40mFiY2RlZjAxMjM0NTY30Dkw
```

Content-Type

Set the value of this parameter to '`application/x-www-form-urlencoded`'.

Request parameters in body

grant_type

(Required) The type of OIDC grant that you want to request.

Must be `authorization_code` or `refresh_token` or `client_credentials`. You can request an access token for a custom scope from the token endpoint under the following conditions:

- You enabled the requested scope in your app client configuration.
- You configured your app client with a client secret.
- You enable client credentials grant in your app client.

client_id

(Optional) The ID of an app client in your user pool. Specify the same app client that authenticated your user.

You must provide this parameter if the client is public and does not have a secret, or with `client_secret` in `client_secret_post` authorization.

client_secret

(Optional) The client secret for the app client that authenticated your user. Required if your app client has a client secret and you did not send an Authorization header.

scope

(Optional) Can be a combination of any custom scopes that are associated with an app client. Any scope that you request must be activated for the app client. If not, Amazon Cognito will ignore it. If the client doesn't request any scopes, the authentication server assigns all custom scopes that you authorized in your app client configuration.

Only used if the `grant_type` is `client_credentials`.

redirect_uri

(Optional) Must be the same `redirect_uri` that was used to get `authorization_code` in /oauth2/authorize.

You must provide this parameter if `grant_type` is `authorization_code`.

refresh_token

(Optional) To generate new access and ID tokens for a user's session, set the value of a refresh_token parameter in your /oauth2/token request to a previously issued refresh token from the same app client.

code

(Optional) The authorization code from an authorization code grant. You must provide this parameter if your authorization request included a grant_type of authorization_code.

code_verifier

(Optional) The arbitrary value that you used to calculate the code_challenge in an authorization code grant request with PKCE.

Example requests with positive responses

Exchanging an authorization code for tokens

Example – POST request

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token&
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWtaTI4M2V10TI40mFiY2RlZjAxMjM0NTY30Dkw

grant_type=authorization_code&
client_id=example23456789&
code=AUTHORIZATION_CODE&
redirect_uri=com.myclientapp://myclient/redirect
```

Example – response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "refresh_token": "eyJj3example",
    "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: client secret in authorization header**Example – POST request**

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWtaTI4M2V10TI40mFiY2RlZjAxMjM0NTY30Dkw

grant_type=client_credentials&
client_id=lexample23456789&
scope=resourceServerIdentifier1/scope1 resourceServerIdentifier2/scope2
```

Example – response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "access_token": "eyJralexample",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Exchanging client credentials for an access token: client secret in request body**Example – POST request**

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
X-Amz-Target: AWSCognitoIdentityProviderService.Client credentials request
User-Agent: USER_AGENT
Accept: /
Accept-Encoding: gzip, deflate, br
Content-Length: 177
Referer: http://auth.example.com/oauth2/token
```

```
Host: auth.example.com
Connection: keep-alive
```

```
grant_type=client_credentials&client_id=1example23456789&scope=my_resource_server_identifier%2F
```

Example – response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Date: Tue, 05 Dec 2023 16:11:11 GMT
x-amz-cognito-request-id: 829f4fe2-a1ee-476e-b834-5cd85c03373b

{
    "access_token": "eyJra12345EXAMPLE",
    "expires_in": 3600,
    "token_type": "Bearer"
}
```

Exchanging an authorization code grant with PKCE for tokens

Example – POST request

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWRTaTI4M2V10TI40mFiY2RlZjAxMjM0NTY30Dkw

grant_type=authorization_code&
client_id=1example23456789&
code=AUTHORIZATION_CODE&
code_verifier=CODE_VERIFIER&
redirect_uri=com.myclientapp://myclient/redirect
```

Example – response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "refresh_token": "eyJj3example",
    "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

Example – POST request

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWtaTI4M2V10TI40mFiY2R1ZjAxMjM0NTY30Dkw

grant_type=refresh_token&
client_id=1example23456789&
refresh_token=eyJj3example
```

Example – response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "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

Example – 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.

App client doesn't have read access to all [attributes](#) in the requested scope. For example, your app requests the `email` scope and your app client can read the `email` attribute, but not `email_verified`.

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` or `client_credentials`.

The user attributes endpoint

Where OIDC issues ID tokens that contain user attributes, OAuth 2.0 implements the `/oauth2/userInfo` endpoint. An authenticated user or client receives an access token with a scopes

claim. This claim determines the attributes that the authorization server should return. When an application presents an access token to the `userInfo` endpoint, the authorization server returns a response body that contains the user attributes that are within the boundaries set by the access token scopes. Your application can retrieve information about a user from the `userInfo` endpoint as long as it holds a valid access token with at least an `openid` scope claim.

The `userInfo` endpoint is an OpenID Connect (OIDC) [userInfo endpoint](#). It responds with user attributes when service providers present access tokens that your [token endpoint](#) issued. The scopes in your user's access token define the user attributes that the `userInfo` endpoint returns in its response. The `openid` scope must be one of the access token claims.

Amazon Cognito issues access tokens in response to user pools API requests like [InitiateAuth](#). Because they don't contain any scopes, the `userInfo` endpoint doesn't accept these access tokens. Instead, you must present access tokens from your token endpoint.

Your OAuth 2.0 third-party identity provider (IdP) also hosts a `userInfo` endpoint. When your user authenticates with that IdP, Amazon Cognito silently exchanges an authorization code with the IdP token endpoint. Your user pool passes the IdP access token to authorize retrieval of user information from the IdP `userInfo` endpoint.

GET /oauth2/userInfo

Your app 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](#)
- [Example – request](#)
- [Example – positive response](#)
- [Example negative responses](#)

Request parameters in header

Authorization: Bearer <access_token>

Pass the access token in the authorization header field.

Required.

Example – request

```
GET /oauth2/userInfo HTTP/1.1
Content-Type: application/x-amz-json-1.1
Authorization: Bearer eyJra12345EXAMPLE
User-Agent: [User agent]
Accept: /*
Host: auth.example.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Example – positive response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: [Integer]
Date: [Timestamp]
x-amz-cognito-request-id: [UUID]
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
Server: Server
Connection: keep-alive
{
    "sub": "[UUID]",
    "email_verified": "true",
    "custom:mycustom1": "CustomValue",
    "phone_number_verified": "true",
    "phone_number": "+12065551212",
    "email": "bob@example.com",
    "username": "bob"
}
```

For a list of OIDC claims, see [Standard Claims](#). Currently, Amazon Cognito returns the values for `email_verified` and `phone_number_verified` as strings.

Example negative responses

Example – bad 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, it includes an unsupported parameter value, or it is otherwise malformed.

Example – bad 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 it's invalid.

The token revocation endpoint

Users who hold a refresh token in their session have something similar to a browser cookie. They can renew their existing session as long as the refresh token is valid. Instead of prompting a user to sign in after their ID or access token expires, your application can use the refresh token to get new, valid tokens. However, you might externally determine that a user's session should be ended, or the user might elect to forget their current session. At that point, you can revoke that refresh token so that they can no longer persist their session.

The /oauth2/revoke endpoint revokes a user's access token that Amazon Cognito initially issued with the refresh token that you provide. This endpoint also revokes the refresh token itself and all subsequent access and identity tokens from the same refresh token. After the endpoint revokes the tokens, you can't use the revoked access tokens to access APIs that Amazon Cognito tokens authenticate.

POST /oauth2/revoke

The /oauth2/revoke 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 your app client has a client secret, the application 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`

(Required) The refresh token that the client wants to revoke. The request also revokes all access tokens that Amazon Cognito issued with this refresh token.

Required.

`client_id`

(Optional) The app client ID for the token that you want to revoke.

Required if the client is public and doesn't have a secret.

Revocation request examples

This revocation request revokes a refresh token for an app client that has no client secret. Note the `client_id` parameter in the request body.

```
POST /oauth2/revoke HTTP/1.1
Host: https://mydomain.auth.us-east-1.amazoncognito.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
```

```
token=2YotnFZFEjr1zCsicMWpAA&  
client_id=djc98u3jiedmi283eu928
```

This revocation request revokes a refresh token for an app client that *has* a client secret. Note the Authorization header that contains an encoded client ID and client secret, but no `client_id` in the request body.

```
POST /oauth2/revoke HTTP/1.1  
Host: https://mydomain.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, in some cases, an `error_description` field.

Endpoint errors

- If the token isn't present in the request or if the feature is disabled for the app client, you receive an HTTP 400 and error `invalid_request`.
- If the token that Amazon Cognito sent in the revocation request isn't a refresh token, you receive an HTTP 400 and error `unsupported_token_type`.
- If the client credentials aren't valid, you receive an HTTP 401 and error `invalid_client`.
- If the token has been revoked or if the client submitted a token that isn't valid, you receive an HTTP 200 OK.

The IdP SAML assertion endpoint

The `/saml2/idpreponse` receives SAML assertions. In service-provider-initiated (SP-initiated) sign-in, your application doesn't interact directly with this endpoint—your SAML 2.0 identity provider (IdP) redirects your user here with their SAML response. For SP-initiated sign-in, configure your IdP with the path to your `saml2/idpreponse` as the assertion consumer service (ACS) URL. For more information about session initiation, see [SAML session initiation in Amazon Cognito user pools](#).

In IdP-initiated sign-in, invoke requests to this endpoint in your application after you sign in user with your SAML 2.0 provider. Your users sign in with your IdP in their browser, then your application collects the SAML assertion and submits it to this endpoint. You must submit SAML assertions in the body of a HTTP POST request over HTTPS. The body of your POST request must be a `SAMLResponse` parameter and a `Relaystate` parameter. For more information, see [Using IdP-initiated SAML sign-in](#).

The `saml2/idpreponse` endpoint can accept SAML assertions of up to 100,000 characters in length.

POST /saml2/idpreponse

To use the `/saml2/idpreponse` endpoint in an IdP-initiated sign-in, generate a POST request with parameters that provide your user pool with information about your user's session.

- The app client that they want to sign in to.
- The callback URL that they want to end up at.
- The OAuth 2.0 scopes that they want to request in your user's access token.
- The IdP that initiated the sign-in request.

IdP-initiated request body parameters

SAMLResponse

A Base64-encoded SAML assertion from an IdP associated with a valid app client and IdP configuration in your user pool.

RelayState

A `RelayState` parameter contains the request parameters that you would otherwise pass to the `oauth2/authorize` endpoint. For detailed information about these parameters, see [Authorize endpoint](#).

response_type

The OAuth 2.0 grant type.

client_id

The app client ID.

redirect_uri

The URL where the authentication server redirects the browser after Amazon Cognito authorizes the user.

identity_provider

The name of the identity provider where you want to redirect your user.

idp_identifier

The identifier of the identity provider where you want to redirect your user.

scope

The OAuth 2.0 scopes that you want your user to request from the authorization server.

Example requests with positive responses

Example – POST request

The following request is for an authorization code grant for a user from IdP MySAMLIdP in app client 1example23456789. The user redirects to <https://www.example.com> with their authorization code, which can be exchanged for tokens that include an access token with the OAuth 2.0 scopes openid, email, and phone.

```
POST /saml2/idpreponse HTTP/1.1
User-Agent: USER_AGENT
Accept: */*
Host: example.auth.us-east-1.amazoncognito.com
Content-Type: application/x-www-form-urlencoded

SAMLResponse=[Base64-encoded SAML assertion]&RelayState=identity_provider
%3DMySAMLIdP%26client_id%3D1example23456789%26redirect_uri%3Dhttps%3A%2F
%2Fwww.example.com%26response_type%3Dcode%26scope%3Demail%2Bopenid%2Bphone
```

Example – response

The following is the response to the previous request.

```
HTTP/1.1 302 Found
Date: Wed, 06 Dec 2023 00:15:29 GMT
Content-Length: 0
x-amz-cognito-request-id: 8aba6eb5-fb54-4bc6-9368-c3878434f0fb
```

Location: [https://www.example.com?code=\[Authorization code\]](https://www.example.com?code=[Authorization code])

OAuth 2.0 grants

The Amazon Cognito user pool OAuth 2.0 authorization server issues tokens in response to three types of OAuth 2.0 [authorization grants](#). You can set the supported grant types for each app client in your user pool. You can't enable *client credentials* grants in the same app client as either *implicit* or *authorization code* grants. Requests for implicit and authorization code grants begin at your [Authorize endpoint](#) and requests for client credentials grants start at your [Token endpoint](#).

Authorization code grant

In response to your successful authentication request, the authorization server appends an authorization code in a `code` parameter to your callback URL. You must then exchange the code for ID, access, and refresh tokens with the [Token endpoint](#). To request an authorization code grant, set `response_type` to `code` in your request. For an example request, see [Authorization code grant](#).

The authorization code grant is the most secure form of authorization grant. It doesn't show token contents directly to your users. Instead, your app is responsible for retrieving and securely storing your user's tokens. In Amazon Cognito, an authorization code grant is the only way to get all three token types—ID, access, and refresh—from the authorization server. You can also get all three token types from authentication through the Amazon Cognito user pools API, but the API doesn't issue access tokens with scopes other than `aws.cognito.signin.user.admin`.

Implicit grant

In response to your successful authentication request, the authorization server appends an access token in an `access_token` parameter, and an ID token in an `id_token` parameter, to your callback URL. An implicit grant requires no additional interaction with the [Token endpoint](#). To request an implicit grant, set `response_type` to `token` in your request. The implicit grant only generates an ID and access token. For an example request, see [Token grant without openid scope](#).

The implicit grant is a legacy authorization grant. Unlike with the authorization code grant, users can intercept and inspect your tokens. To prevent token delivery through implicit grant, configure your app client to support authorization code grant only.

Client credentials

Client credentials is an authorization-only grant for machine-to-machine access. To receive a client credentials grant, bypass the [Authorize endpoint](#) and generate a request directly to the [Token endpoint](#). Your app client must have a client secret and support client credentials grants only. In response to your successful request, the authorization server returns an access token.

The access token from a client credentials grant is an authorization mechanism that contains OAuth 2.0 scopes. Typically, the token contains custom scope claims that authorize HTTP operations to access-protected APIs. For more information, see [Scopes, M2M, and APIs with resource servers](#).

Client credentials grants add costs to your AWS bill. For more information, see [Amazon Cognito Pricing](#).

For more perspective on these grants and their implementation, see [How to use OAuth 2.0 in Amazon Cognito: Learn about the different OAuth 2.0 grants](#) in the *AWS Security Blog*.

Using PKCE in authorization code grants

Amazon Cognito supports Proof Key for Code Exchange (PKCE) authentication in authorization code grants. PKCE is an extension to the OAuth 2.0 authorization code grant for public clients. PKCE guards against the redemption of intercepted authorization codes.

How Amazon Cognito uses PKCE

To start authentication with PKCE, your application must generate a unique string value. This string is the code verifier, a secret value that Amazon Cognito uses to compare the client requesting the initial authorization grant to the client exchanging the authorization code for tokens.

Your app must apply an SHA256 hash to the code verifier string and encode the result to base64. Pass the hashed string to the [Authorize endpoint](#) as a `code_challenge` parameter in the request body. When your app exchanges the authorization code for tokens, it must include the code verifier string in plaintext as a `code_verifier` parameter in the request body to the [Token endpoint](#). Amazon Cognito performs the same hash-and-encode operation on the code verifier. Amazon Cognito only returns ID, access, and refresh tokens if it determines that the code verifier results in the same code challenge that it received in the authorization request.

To implement Authorization Grant Flow with PKCE

1. Open the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool. If you create a user pool, you will be prompted to set up an app client and configure managed login during the wizard.
 - a. If you create a new user pool, set up an app client and configure managed login during the guided setup.
 - b. If you configure an existing user pool, add a [domain](#) and a [public app client](#), if you haven't already.
4. Generate a random alphanumeric string, typically a universally unique identifier ([UUID](#)), to create a code challenge for the PKCE. This string is the value of the `code_verifier` parameter that you will submit in your request to the [Token endpoint](#).
5. Hash the `code_verifier` string with the SHA256 algorithm. Encode the result of the hashing operation to base64. This string is the value of the `code_challenge` parameter that you will submit in your request to the [Authorize endpoint](#).

The following Python example generates a `code_verifier` and calculates the `code_challenge`:

```
#!/usr/bin/env python3

import random
from base64 import urlsafe_b64encode
from hashlib import sha256
from string import ascii_letters
from string import digits

# use a cryptographically strong random number generator source
rand = random.SystemRandom()

code_verifier = ''.join(rand.choices(ascii_letters + digits, k=128))
code_verifier_hash = sha256(code_verifier.encode()).digest()
code_challenge = urlsafe_b64encode(code_verifier_hash).decode().rstrip('=')

print(f"code challenge: {code_challenge}")
print(f"code verifier: {code_verifier}")
```

The following is an example output from the Python script:

```
code challenge: Eh0mg-0Zv7BAyo-tdv_vYamx1bo0YDulDklyXoMDtLg
code verifier: 9D-aW_iygXrgQcWJd0y0tNVMPXSChIc2xceDhvYVdGLCBk-
JWTmBNjvKSd0rjTTYaz0FbUmrFERrjWx6oKtK2b6z_x4_gHBDlr4K1mRFGyE8yA-05-_v7Dxf3EIYJH
```

6. Complete managed login sign-in with an authorization code grant request with PKCE. The following is an example URL:

```
https://mydomain.us-east-1.amazonaws.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com&code_challenge=Eh0mg-0Zv7BAyo-

```

7. Collect the authorization code and redeem it for tokens with the token endpoint. The following is an example request:

```
POST /oauth2/token HTTP/1.1
Host: mydomain.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 296

redirect_uri=https%3A%2F%2Fwww.example.com&
client_id=1example23456789&
code=7378f445-c87f-400c-855e-0297d072ff03&
grant_type=authorization_code&
code_verifier=9D-aW_iygXrgQcWJd0y0tNVMPXSChIc2xceDhvYVdGLCBk-
JWTmBNjvKSd0rjTTYaz0FbUmrFERrjWx6oKtK2b6z_x4_gHBDlr4K1mRFGyE8yA-05-_v7Dxf3EIYJH
```

8. Review the response. It will contain ID, access, and refresh tokens. For more information about using Amazon Cognito user pool tokens, see [Understanding user pool JSON web tokens \(JWTs\)](#).

Managed login and federation error responses

A sign-in process in managed login or federated sign-in might return an error. The following are some conditions that can cause authentication to end with an error.

- A user performs an operation that your user pool can't fulfill.
- A Lambda trigger doesn't respond with expected syntax.

- Your identity provider (IdP) returns an error.
- Amazon Cognito couldn't validate attribute information that your user provided.
- Your IdP didn't send claims that map to required attributes.

When Amazon Cognito encounters an error, it communicates it in one of the following ways.

1. Amazon Cognito sends a redirect URL with the error in the request parameters.
2. Amazon Cognito displays an error in managed login.

Errors that Amazon Cognito appends to request parameters have the following format.

```
https://<Callback URL>/?error_description=error+description&error=error+name
```

When you help your users submit error information when they can't perform an operation, request that they capture the URL *and* the text or a screenshot of the page.

 **Note**

Amazon Cognito error descriptions are not fixed strings and you shouldn't use logic that relies on a fixed pattern or format.

OIDC and social identity provider error messages

Your identity provider might return an error. When an OIDC or OAuth 2.0 IdP returns an error that conforms to standards, Amazon Cognito redirects your user to the callback URL and adds the provider error response to error request parameters. Amazon Cognito adds the provider name and HTTP error code to the existing error strings.

The following URL is an example redirect from an IdP that returned an error to Amazon Cognito.

```
https://www.amazon.com/?error_description=LoginWithAmazon+Error++400+invalid_request
+The+request+is+missing+a+required+parameter+%3A+client_secret&error=invalid_request
```

Because Amazon Cognito only returns what it receives from a provider, your user might see a subset of this information.

When your user encounters an issue with initial sign-in through your IdP, the IdP delivers any error messages directly to your user. Amazon Cognito relays an error message to your user when it generates a request to your IdP to validate your user's session. Amazon Cognito relays OAuth and OIDC IdP error messages from the following endpoints.

`/token`

Amazon Cognito exchanges an IdP authorization code for an access token.

`/.well-known/openid-configuration`

Amazon Cognito discovers the path to your issuer endpoints.

`/.well-known/jwks.json`

To verify your user's JSON Web Tokens (JWTs), Amazon Cognito discovers the JSON Web Keys (JWKs) that your IdP uses to sign tokens.

Because Amazon Cognito doesn't initiate outbound sessions to SAML 2.0 providers that might return HTTP errors, your users' errors during a session with a SAML 2.0 IdP don't include this form of provider error message.

Amazon Cognito identity pools

An Amazon Cognito identity pool is a directory of federated identities that you can exchange for AWS credentials. Identity pools generate temporary AWS credentials for the users of your app, whether they've signed in or you haven't identified them yet. With AWS Identity and Access Management (IAM) roles and policies, you can choose the level of permission that you want to grant to your users. Users can start out as guests and retrieve assets that you keep in AWS services. Then they can sign in with a third-party identity provider to unlock access to assets that you make available to registered members. The third-party identity provider can be a consumer (social) OAuth 2.0 provider like Apple or Google, a custom SAML or OIDC identity provider, or a custom authentication scheme, also called a *developer provider*, of your own design.

Features of Amazon Cognito identity pools

Sign requests for AWS services

[Sign API requests](#) to AWS services like Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB. Analyze user activity with services like Amazon Pinpoint and Amazon CloudWatch.

Filter requests with resource-based policies

Exercise granular control over user access to your resources. Transform user claims into [IAM session tags](#), and build IAM policies that grant resource access to distinct subsets of your users.

Assign guest access

For your users who haven't signed in yet, configure your identity pool to generate AWS credentials with a narrow scope of access. Authenticate users through a single sign-on provider to elevate their access.

Assign IAM roles based on user characteristics

Assign a single IAM role to all of your authenticated users, or choose the role based on the claims of each user.

Accept a variety of identity providers

Exchange an ID or access token, a user pool token, a SAML assertion, or a social-provider OAuth token for AWS credentials.

Validate your own identities

Perform your own user validation and use your developer AWS credentials to issue credentials for your users.

You might already have an Amazon Cognito user pool that provides authentication and authorization services to your app. You can set up your user pool as an identity provider (IdP) to your identity pool. When you do, your users can authenticate through your user pool IdPs, consolidate their claims into a common OIDC identity token, and exchange that token for AWS credentials. Your user can then present their credentials in a signed request to your AWS services.

You can also present authenticated claims from any of your identity providers directly to your identity pool. Amazon Cognito customizes user claims from SAML, OAuth, and OIDC providers into an [AssumeRoleWithWebIdentity](#) API request for short-term credentials.

Amazon Cognito user pools are like OIDC identity providers to your SSO-enabled apps. Identity pools act as an AWS identity provider to any app with resource dependencies that work best with IAM authorization.

Amazon Cognito identity pools support the following identity providers:

- Public providers: [Setting up Login with Amazon as an identity pools IdP](#), [Setting up Facebook as an identity pools IdP](#), [Setting up Google as an identity pool IdP](#), [Setting up Sign in with Apple as an identity pool IdP](#), Twitter.
- [Amazon Cognito user pools](#)
- [Setting up an OIDC provider as an identity pool IdP](#)
- [Setting up a SAML provider as an identity pool IdP](#)
- [Developer-authenticated identities](#)

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

- [Identity pools console overview](#)
- [Identity pools authentication flow](#)

- [IAM roles](#)
- [Security best practices for Amazon Cognito identity pools](#)
- [Using attributes for access control](#)
- [Using role-based access control](#)
- [Getting credentials](#)
- [Accessing AWS services with temporary credentials](#)
- [Identity pools third-party identity providers](#)
- [Developer-authenticated identities](#)
- [Switching unauthenticated users to authenticated users](#)

Identity pools console overview

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 identifiers linked to your external identity providers.

One way to understand the features and options of identity pools is to create one in the Amazon Cognito console. You can explore the effect of different settings on authentication flows, role-based and attribute-based access control, and guest access. From there, you can proceed to later chapters in this guide and add the appropriate components to your application so that you can implement identity pool authentication.

Topics

- [Create an identity pool](#)
- [User IAM roles](#)
- [Authenticated and unauthenticated identities](#)
- [Activate or deactivate guest access](#)
- [Change the role associated with an identity type](#)
- [Edit identity providers](#)
- [Delete an identity pool](#)
- [Delete an identity from an identity pool](#)
- [Using Amazon Cognito Sync with identity pools](#)

Create an identity pool

To create a new identity pool in the console

1. Sign in to the [Amazon Cognito console](#) and select **Identity pools**.
2. Choose **Create identity pool**.
3. In **Configure identity pool trust**, choose to set up your identity pool for **Authenticated access**, **Guest access**, or both.
 - If you chose **Authenticated access**, select one or more **Identity types** that you want to set as the source of authenticated identities in your identity pool. If you configure a **Custom developer provider**, you can't modify or delete it after you create your identity pool.
4. In **Configure permissions**, choose a default IAM role for authenticated or guest users in your identity pool.
 - a. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
 - b. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see [Role trust and permissions](#).
5. In **Connect identity providers**, enter the details of the identity providers (IdPs) that you chose in **Configure identity pool trust**. You might be asked to provide OAuth app client information, choose a Amazon Cognito user pool, choose an IAM IdP, or enter a custom identifier for a developer provider.
 - a. Choose the **Role settings** for each IdP. You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With a Amazon Cognito user pool IdP, you can also **Choose role with preferred_role in tokens**. For more information about the `cognito:preferred_role` claim, see [Assigning precedence values to groups](#).

- i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
- b. Configure **Attributes for access control** for each IdP. Attributes for access control maps user claims to [principal tags](#) that Amazon Cognito applies to their temporary session. You can build IAM policies to filter user access based on the tags that you apply to their session.
 - i. To apply no principal tags, choose **Inactive**.
 - ii. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - iii. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
6. In **Configure properties**, enter a **Name** under **Identity pool name**.
 7. Under **Basic (classic) authentication**, choose whether you want to **Activate basic flow**. With the basic flow active, you can bypass the role selections you made for your IdPs and call [AssumeRoleWithWebIdentity](#) directly. For more information, see [Identity pools authentication flow](#).
 8. Under **Tags**, choose **Add tag** if you want to apply [tags](#) to your identity pool.
 9. In **Review and create**, confirm the selections that you made for your new identity pool. Select **Edit** to return to the wizard and change any settings. When you're done, select **Create identity pool**.

User IAM roles

An IAM role defines the permissions for your users to access AWS resources, like [Amazon Cognito Sync](#). 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](#).

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 third-party identity providers](#).
- To configure your own backend authentication process, see [Developer-authenticated identities](#).

Activate or deactivate guest access

Amazon Cognito identity pools guest access (unauthenticated identities) provides 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 activate access for unauthenticated identities. To learn more, see [Getting started with Amazon Cognito identity pools](#).

To update guest access in an identity pool

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Guest access**. In an identity pool that doesn't currently support guest access, **Status** is **Inactive**.
 - a. If **Guest access** is **Active** and you want to deactivate it, select **Deactivate**.
 - b. If **Guest access** is **Inactive** and you want to activate it, select **Edit**.
 - Choose a default IAM role for guest users in your identity pool.
 - A. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
 - B. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents

evidence that the request originated from an authenticated user in your specific identity pool. For more information, see [Role trust and permissions](#).

- C. Select **Save changes**.
- D. To activate guest access, select **Activate** in the **User access** tab.

Change the role associated with an identity type

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 that dictates which AWS services that role can access. When Amazon Cognito receives a request, the service determines the identity type, determines the role assigned to that identity type, and uses 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](#).

To change the identity pool default authenticated or unauthenticated role

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Guest access** or **Authenticated access**. In an identity pool that isn't currently configured for that access type, **Status** is **Inactive**. Select **Edit**.
4. Choose a default IAM role for guest or authenticated users in your identity pool.
 - a. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
 - b. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon

Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see [Role trust and permissions](#).

5. Select **Save changes**.

Edit identity providers

If you allow your users to authenticate using consumer identity providers (for example, Amazon Cognito user pools, Login with Amazon, Sign in with Apple, Facebook, or 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 [Using role-based access control](#).

 **Warning**

Changing the linked IdP application ID in your identity pool prevents existing users from authenticating with that identity pool. For more information, see [Identity pools third-party identity providers](#).

To update an identity pool identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Identity providers**. Choose the identity provider that you want to edit. If you want to add a new IdP, select **Add identity provider**.
 - If you chose **Add identity provider**, choose one of the **Identity types** that you want to add.
4. To change the application ID, choose **Edit in Identity provider information**.
5. To change the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, choose **Edit in Role settings**.

- You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With a Amazon Cognito user pool IdP, you can also **Choose role with preferred_role in tokens**. For more information about the `cognito:preferred_role` claim, see [Assigning precedence values to groups](#).
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
6. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, choose **Edit** in **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
 7. Select **Save changes**.

Delete an identity pool

You can't undo identity pool deletion. After you delete an identity pool, all apps and users that depend on it stop working.

To delete an identity pool

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select the radio button next to the identity pool that you want to delete.
2. Select **Delete**.
3. Enter or paste the name of your identity pool and select **Delete**.

Warning

When you select 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 using the identity pool to stop working.

Delete an identity from an identity pool

When you delete an identity from an identity pool, you remove the identifying information that Amazon Cognito has stored for that federated user. When your user requests credentials again, they receive a new identity ID if your identity pool still trusts their identity provider. You can't undo this operation.

To delete an identity

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **Identity browser** tab.
3. Select the check boxes next to the identities that you want to delete and choose **Delete**. Confirm that you want to delete the identities and choose **Delete**.

Using Amazon Cognito Sync with identity pools

Amazon Cognito Sync is an AWS service and client library that makes it possible to sync application-related user data across devices. Amazon Cognito Sync can synchronize user profile data across mobile devices and the web without using your own backend. The client libraries cache data locally so that your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data. If you set up push sync, you can notify other devices immediately that an update is available.

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 isn't saved until you select **Synchronize** in the console. The change isn't visible to the end user until the identity calls **Synchronize**. The data being synchronized

from other devices for individual identities is visible once you refresh the list datasets page for a particular identity.

Create a dataset for an identity

Amazon Cognito Sync associates a dataset with one identity. You can populate your dataset with identifying information about the user that the identity represents, then sync that information to all of your user's devices.

To add a dataset and dataset records to an identity

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **Identity browser** tab.
3. Select the identity that you want to edit.
4. In **Datasets**, choose **Create dataset**.
5. Enter a **Dataset name** and select **Create dataset**.
6. If you want to add records to your dataset, choose your dataset from identity details. In **Records**, select **Create record**.
7. Enter a **Key** and **Value** for your record. Choose **Confirm**. Repeat to add more records.

Delete a dataset associated with an identity

To delete a dataset and its records from an identity

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **Identity browser** tab.
3. Select the identity that contains the dataset that you want to delete.
4. In **Datasets**, choose the radio button next to the dataset that you want to delete.
5. Select **Delete**. Review your choice and select **Delete** again.

Bulk publish data

Bulk publish can be used to export data already stored in your Amazon Cognito Sync store to a Amazon Kinesis stream. For instructions on how to bulk publish all of your streams, see [Implementing Amazon Cognito Sync streams](#).

Activate push synchronization

Amazon Cognito automatically tracks the association between identity and devices. Using the push sync feature, you can make sure that every instance of a given identity is notified when identity data changes. Push sync makes it so that, whenever the dataset changes for an identity, all devices associated with that identity receive a silent push notification informing them of the change.

You can activate push sync in the Amazon Cognito console.

To activate push synchronization

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **Identity pool properties** tab.
3. In **Push synchronization**, select **Edit**.
4. Select **Activate push synchronization with your identity pool**.
5. Choose one of the Amazon Simple Notification Service (Amazon SNS) **Platform applications** that you created in the current AWS Region. Amazon Cognito publishes push notifications to your platform application. Select **Create platform application** to navigate to the Amazon SNS console and create a new one.
6. To publish to your platform application, Amazon Cognito assumes an IAM role in your AWS account. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
7. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see [Role trust and permissions](#).
8. Select **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 [Implementing Amazon Cognito Sync streams](#).

Set up Amazon Cognito Events

Amazon Cognito Events allows you to run 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 [Customizing workflows with Amazon Cognito Events](#).

To learn more about AWS Lambda, see [AWS Lambda](#).

Identity pools 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 lifecycle of an identity inside your identity pool.

External provider authflow

A user authenticating with Amazon Cognito goes 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](#) creates 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](#).

Identity pools accept the following artifacts from providers:

Provider	Authentication artifact
Amazon Cognito user pool	ID token

Provider	Authentication artifact
OpenID Connect (OIDC)	ID token
SAML 2.0	SAML assertion
Social provider	Access token

Enhanced (simplified) authflow

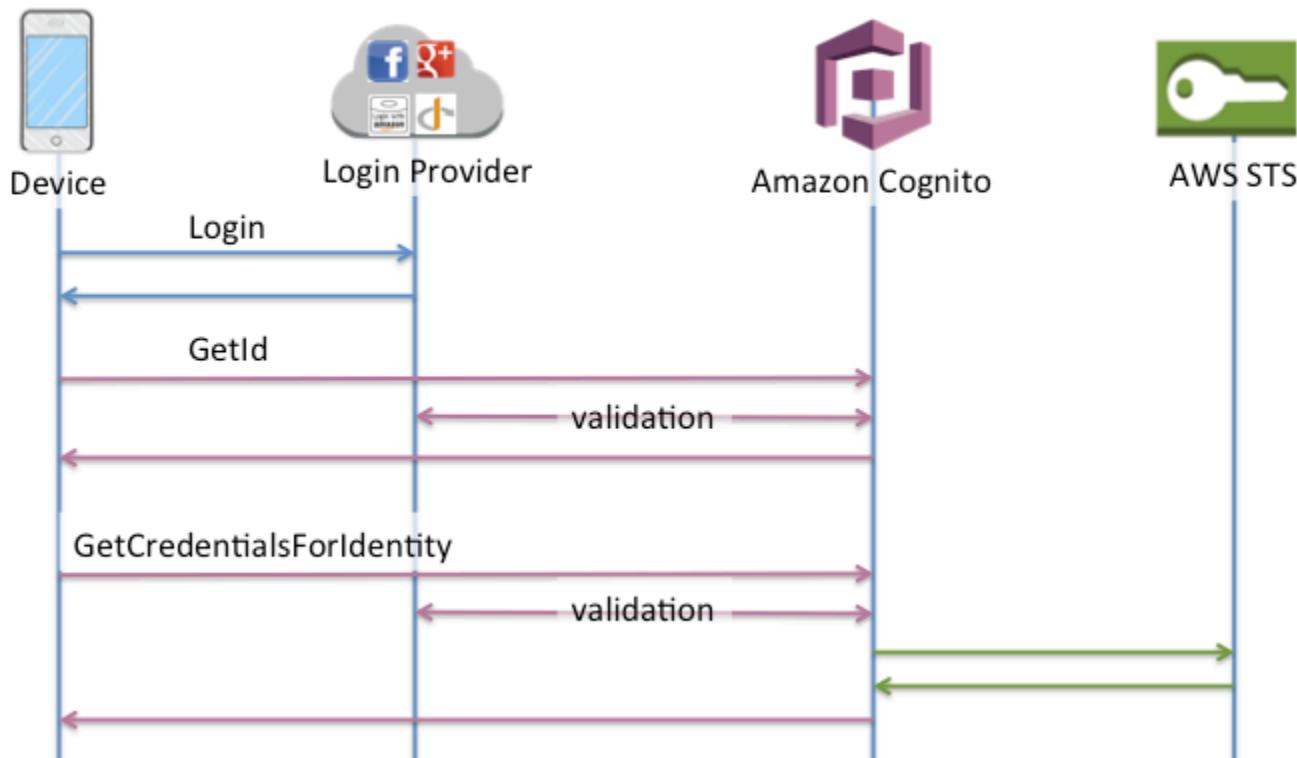
When you use the enhanced authflow, your app first presents a proof of authentication from an authorized Amazon Cognito user pool or third-party identity provider in a [GetId](#) request.

1. Your application presents a proof of authentication—a JSON web token or a SAML assertion—from an authorized Amazon Cognito user pool or third-party identity provider in a [GetID](#) request.
2. Your identity pool returns an identity ID.
3. Your application combines the identity ID with the same proof of authentication in a [GetCredentialsForIdentity](#) request.
4. Your identity pool returns AWS credentials.
5. Your application signs AWS API requests with the temporary credentials.

Enhanced authentication manages the logic of IAM role selection and credentials retrieval in your identity pool configuration. You can configure your identity pool to select a default role, to apply attribute-based access control (ABAC) or role-based access control (RBAC) principles to role selection. The AWS credentials from enhanced authentication are valid for one hour.

Order of operations in Enhanced authentication

1. GetId
2. GetCredentialsForIdentity



Basic (classic) authflow

When you use the basic authflow,

1. Your application presents a proof of authentication—a JSON web token or a SAML assertion—from an authorized Amazon Cognito user pool or third-party identity provider in a [GetID](#) request.
2. Your identity pool returns an identity ID.
3. Your application combines the identity ID with the same proof of authentication in a [GetOpenIdToken](#) request.
4. GetOpenIdToken returns a new OAuth 2.0 token that is issued by your identity pool.
5. Your application presents the new token in an [AssumeRoleWithWebIdentity](#) request.
6. AWS Security Token Service AWS STS) returns AWS credentials.
7. Your application signs AWS API requests with the temporary credentials.

The basic workflow gives you more granular control over the credentials that you distribute to your users. The **GetCredentialsForIdentity** request of the enhanced authflow requests a role based on the contents of an access token. The **AssumeRoleWithWebIdentity** request in

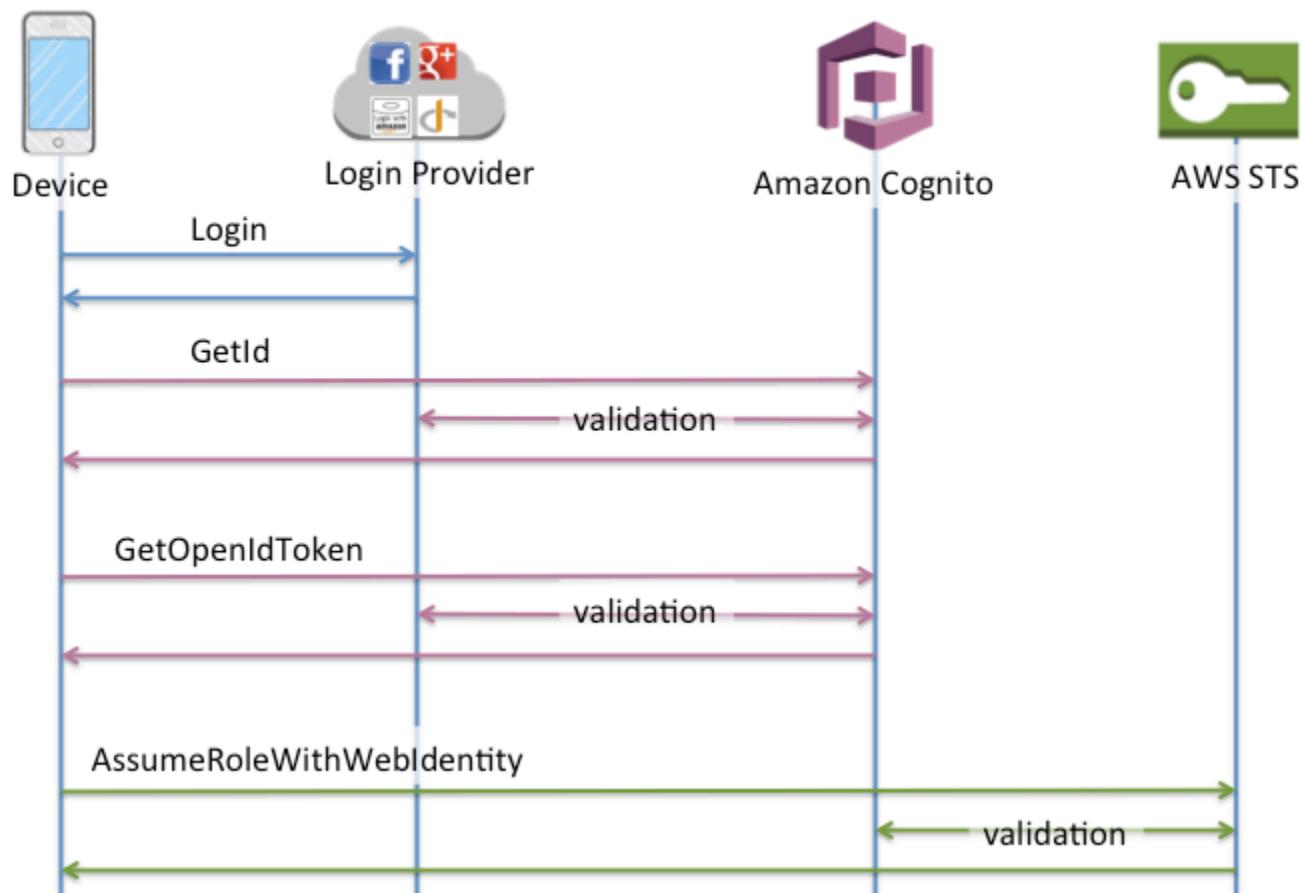
the classic workflow grants your app a greater ability to request credentials for any AWS Identity and Access Management role that you have configured with a sufficient trust policy. You can also request a custom role session duration.

You can sign in with the Basic authflow in user pools that don't have role mappings. This type of identity pool doesn't have a default authenticated or unauthenticated role, and doesn't have role-based or attribute-based access control configured. When you attempt GetOpenIdToken in an identity pool with role mappings, you receive the following error.

Basic (classic) flow is not supported with RoleMappings, please use enhanced flow.

Order of operations in Basic authentication

1. GetById
2. GetOpenIdToken
3. AssumeRoleWithWebIdentity



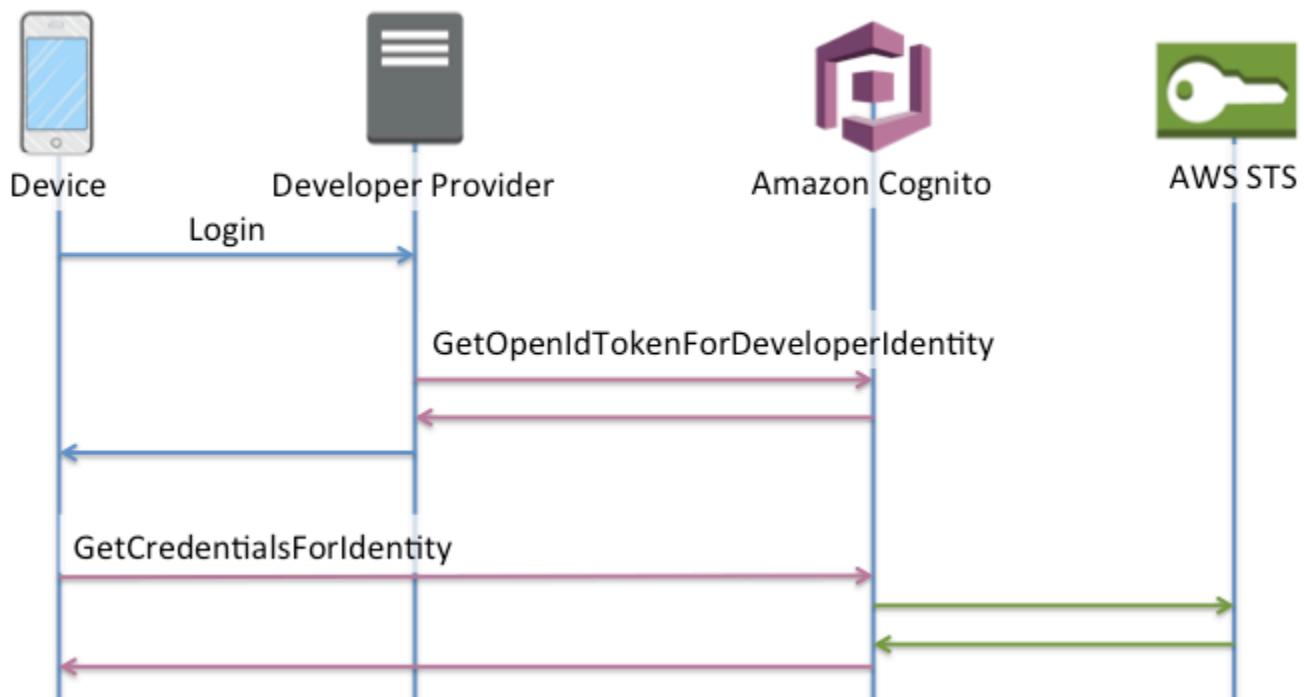
Developer authenticated identities authflow

When using [Developer-authenticated identities](#), the client uses a different authflow that includes 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

Order of operations in Enhanced authentication with a developer provider

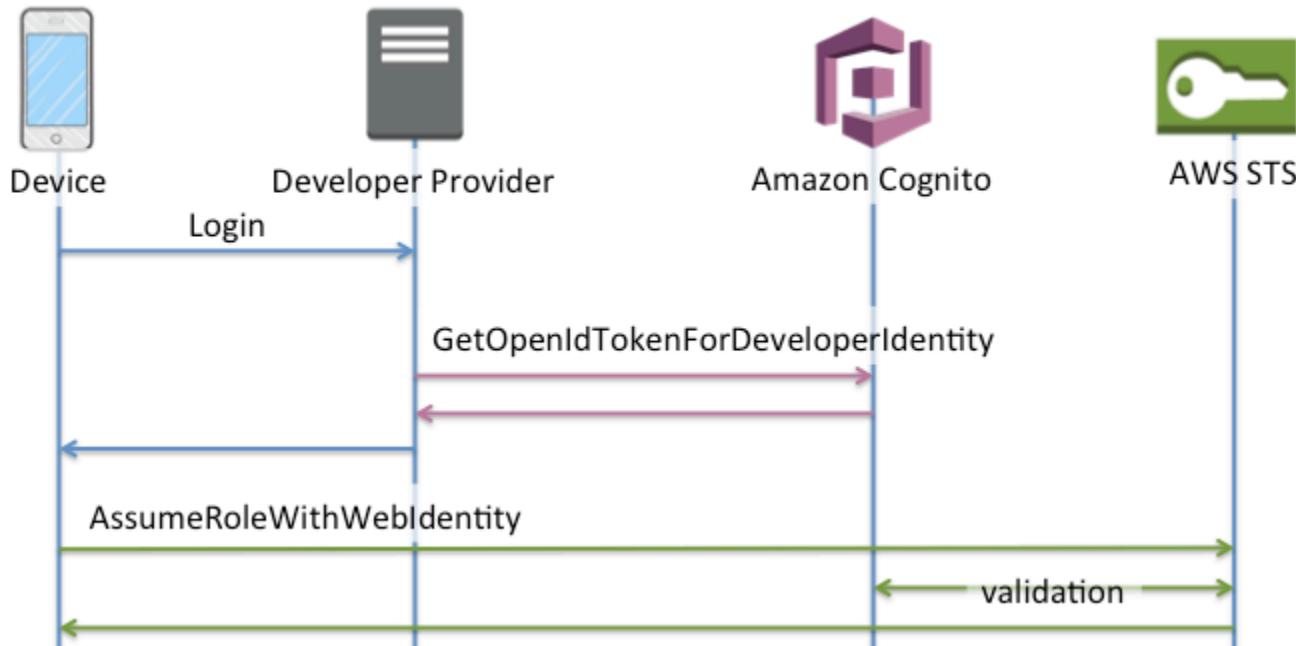
1. Login via Developer Provider (code outside of Amazon Cognito)
2. Validate the user login (code outside of Amazon Cognito)
3. [GetOpenIdTokenForDeveloperIdentity](#)
4. [GetCredentialsForIdentity](#)



Order of operations in Basic authentication with a developer provider

1. Implement logic outside of identity pool to sign in and generate a developer-provider identifier.
2. Retrieve stored server-side AWS credentials.
3. Send developer provider identifier in a [GetOpenIdTokenForDeveloperIdentity](#) API request signed with authorized AWS credentials.

4. Request application credentials with [AssumeRoleWithWebIdentity](#).



Which authflow should I use?

The **enhanced flow** is the most secure choice with the lowest level of developer effort:

- The enhanced flow reduces the complexity, size, and rate of API requests.
- Your application doesn't need to make additional API requests to AWS STS.
- Your identity pool evaluates your users for the IAM role credentials that they should receive. You don't need to embed logic for role selection in your client.

⚠ Important

When you create a new identity pool, don't activate basic (classic) authentication by default, as a best practice. To implement basic authentication, first evaluate the trust relationships of your IAM roles for web identities. Then build the logic for role selection into your client and secure the client against modification by users.

The **basic authentication flow** delegates the logic of IAM role selection to your application. In this flow, Amazon Cognito validates your user's authenticated or unauthenticated session and issues

a token that you can exchange for credentials with AWS STS. Users can exchange the tokens from basic authentication for any IAM roles that trust your identity pool and `amr`, or authenticated/unauthenticated state.

Similarly, understand that **developer authentication** is a shortcut around validation of identity provider authentication. Amazon Cognito trusts the AWS credentials that authorize a [GetOpenIdTokenForDeveloperIdentity](#) request without additional validation of the request contents. Secure the secrets that authorize developer authentication from access by users.

API summary

GetId

The [GetId](#) API call is the first call necessary to establish a new identity in Amazon Cognito.

Unauthenticated access

Amazon Cognito can grant 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 and 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 can also supply tokens (OAuth or OpenID Connect) that identify them in those providers. When used in a call to GetId, Amazon Cognito creates a new authenticated identity or returns the identity already associated with that particular login. Amazon Cognito does this by validating the token with the provider and making sure of the following:

- The token is valid and from the configured provider.
- The token is not expired.
- The token matches the application identifier created with that provider (for example, Facebook app ID).
- The token matches the user identifier.

GetCredentialsForIdentity

The [GetCredentialsForIdentity](#) API can be called after you establish an identity ID. This operation is functionally equivalent to calling [GetOpenIdToken](#), then [AssumeRoleWithWebIdentity](#).

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.

GetOpenIdToken

Make a [GetOpenIdToken](#) API request after you establish an identity ID. Cache identity IDs after your first request, and start subsequent basic (classic) sessions for that identity with `GetOpenIdToken`.

The response to a `GetOpenIdToken` API request is a token that Amazon Cognito generates. You can submit this token as the `WebIdentityToken` parameter in an [AssumeRoleWithWebIdentity](#) request.

Before you submit the OpenID token, verify it in your app. You can use OIDC libraries in your SDK or a library like [aws-jwt-verify](#) to confirm that Amazon Cognito issued the token. The signing key ID, or `kid`, of the OpenID token is one of those listed in the Amazon Cognito Identity [jwks_uri document](#)†. These keys are subject to change. Your function that verifies Amazon Cognito Identity tokens should periodically update its list of keys from the `jwks_uri` document. Amazon Cognito sets the refresh duration in the `jwks_uri` cache-control response header, currently set to a max-age of 30 days.

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 identities or identities that you have deactivated.

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 that you pass in may not be the one that is returned.

Linking logins

If you submit 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 results in a `ResourceConflictException` error response. If a login is merely linked to an existing identity, the identity ID returned from `GetOpenIdToken` is the same as the one that you 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 must update your local cache if this value differs. The providers in the AWS Mobile SDKs or AWS SDK for JavaScript in the Browser perform this operation for you.

GetOpenIdTokenForDeveloperIdentity

The [GetOpenIdTokenForDeveloperIdentity](#) operation replaces the use of [GetId](#) and [GetOpenIdToken](#) from the device when using developer authenticated identities. Because your application signs requests to this API operation with AWS credentials, Amazon Cognito trusts that the user identifier supplied in the request is valid. Developer authentication replaces the token validation that Amazon Cognito performs with external providers.

The payload for this API includes a `logins` map. This map 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 creates a new identity and returns the new identity ID and an OpenID Connect token for that identity. If the user identifier is already linked, Amazon Cognito returns the pre-existing identity ID and an OpenID Connect token. Cache developer identity IDs after your first request, and start subsequent basic (classic) sessions for that identity with `GetOpenIdTokenForDeveloperIdentity`.

The response to a `GetOpenIdTokenForDeveloperIdentity` API request is a token that Amazon Cognito generates. You can submit this token as the `WebIdentityToken` parameter in an `AssumeRoleWithWebIdentity` request.

Before you submit the OpenID Connect token, verify it in your app. You can use OIDC libraries in your SDK or a library like [aws-jwt-verify](#) to confirm that Amazon Cognito issued the token.

The signing key ID, or kid, of the OpenID Connect token is one of those listed in the Amazon Cognito Identity [jwks_uri document](#)†. These keys are subject to change. Your function that verifies Amazon Cognito Identity tokens should periodically update its list of keys from the *jwks_uri* document. Amazon Cognito sets the refresh duration in the *jwks_uri* cache-control response header, currently set to a max-age of 30 days.

Linking logins

As with external providers, supplying additional logins that are not already associated with an identity implicitly links those logins to that identity. If you link an external provider login to an identity, the user can use the external provider authflow with that provider. However, 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 and explicit merging through the [MergeDeveloperIdentities](#) API. With explicit merging, you can mark two identities with user identifiers in your system as a single identity. If you supply the source and destination user identifiers, Amazon Cognito merges them. The next time you request an OpenID Connect token for either user identifier, the same identity id is returned.

AssumeRoleWithWebIdentity

After you have an OpenID Connect token, you can then trade this for temporary AWS credentials through the [AssumeRoleWithWebIdentity](#) API request to AWS Security Token Service (AWS STS).

Because there is no restriction on the number of identities that you can create, it is important to understand the permissions that you're granting to your users. Set up different IAM roles for your application: one for unauthenticated users, and one for authenticated users. The Amazon Cognito console can create default roles when you first set up your identity pool. These roles have effectively no permissions granted. Modify them to meet your needs.

Learn more about [Role trust and permissions](#).

† The default Amazon Cognito Identity *jwks_uri* document contains information about the keys that sign tokens for identity pools in most AWS Regions. The following Regions have different *jwks_uri* documents.

Amazon Cognito Identity JSON web key URIs in other AWS Regions

AWS Region	Path to <i>jwks_uri</i> document
AWS GovCloud (US-West)	https://cognito-identity.us-gov-west-1.amazonaws.com/.well-known/jwks_uri
China (Beijing)	https://cognito-identity.cn-north-1.amazonaws.com.cn/.well-known/jwks_uri
Opt-in Regions like Europe (Milan) and Africa (Cape Town)	https://cognito-identity.<i>Region</i>.amazonaws.com/.well-known/jwks_uri

You can also extrapolate the *jwks_uri* from the issuer or *iss* that you receive in the OpenID token from Amazon Cognito. The OIDC-standard discovery endpoint <issuer>/.well-known/openid-configuration lists a path to the *jwks_uri* for your token.

IAM roles

While creating an identity pool, you're 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. With the IAM role, you can 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 [Using role-based access control](#).

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.

Modify IAM roles to allow or restrict access to other services. To do so, [log in to the IAM Console](#). Then select **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 selecting 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](#)
- [Access policies](#)
- [Role trust and permissions](#)

Set up a trust policy

Amazon Cognito uses 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](#).

The token presented to AWS STS is generated by an identity pool, which translates a user pool, social, or OIDC provider token, or a SAML assertion, to its own token. The identity pool token contains an aud claim that is the identity pool ID.

The following example role trust policy allows the federated service principal `cognito-identity.amazonaws.com` to call the AWS STS API `AssumeRoleWithWebIdentity`. The request will only succeed if the identity pool token in the API request has the following claims.

1. An aud claim of the identity pool ID `us-west-2:abcdefg-1234-5678-910a-0e8443553f95`.
2. An amr claim of `authenticated` that is added when the user has signed in and isn't a guest user.

```
{  
  "Version": "2012-10-17",  
  "Statement": [
```

```
{  
    "Effect": "Allow",  
    "Principal": {  
        "Federated": "cognito-identity.amazonaws.com"  
    },  
    "Action": "sts:AssumeRoleWithWebIdentity",  
    "Condition": {  
        "StringEquals": {  
            "cognito-identity.amazonaws.com:aud": "us-  
west-2:abcdefg-1234-5678-910a-0e8443553f95"  
        },  
        "ForAnyValue:StringLike": {  
            "cognito-identity.amazonaws.com:amr": "authenticated"  
        }  
    }  
}  
]  
}
```

Trust policies for IAM roles in Basic (Classic) authentication

You must apply at least one condition that limits trust policies for roles that you use with identity pools. When you create or update role trust policies for identity pools, IAM returns an error if you try to save your changes without at least one condition key that limits source identities. AWS STS doesn't permit cross-account [AssumeRoleWithWebIdentity](#) operations from identity pools to IAM roles that lack a condition of this type.

This topic includes several conditions that limit source identities for identity pools. For a full list, see [Available keys for AWS web identity federation](#).

In basic, or classic, authentication with an identity pool, you can assume any IAM role with AWS STS if it has the right trust policy. IAM roles for Amazon Cognito identity pools trust the service principal `cognito-identity.amazonaws.com` to assume the role. This configuration is not sufficient to secure your IAM roles against unintended access to resources. Roles of this type must apply an additional condition to the role trust policy. You can't create or modify roles for identity pools without at least one of the following conditions.

cognito-identity.amazonaws.com:aud

Restricts the role to operations from one or more identity pools. Amazon Cognito indicates the source identity pool in the `aud` claim in the identity pool token.

cognito-identity.amazonaws.com:amr

Restricts the role to either authenticated or unauthenticated (guest) users. Amazon Cognito indicates the authentication state in the `amr` claim in the identity pool token.

cognito-identity.amazonaws.com:sub

Restricts the role to one or more users by [UUID](#). This UUID is the user's identity ID in the identity pool. This value isn't the sub value from the user's original identity provider. Amazon Cognito indicates this UUID in the `sub` claim in the identity pool token.

Enhanced-flow authentication requires that the IAM role be in the same AWS account as the identity pool, but this isn't the case in basic authentication.

Additional considerations apply to Amazon Cognito identity pools that assume [cross-account](#) IAM roles. The trust policies of those roles must accept the `cognito-identity.amazonaws.com` service principal *and* must contain the specific `cognito-identity.amazonaws.com:aud` condition. To prevent unintended access to your AWS resources, the `aud` condition key restricts the role to users from the identity pools in the condition value.

The token that an identity pool issues for an identity contains information about the originating AWS account of the identity pool. When you present an identity pool token in an [AssumeRoleWithWebIdentity](#) API request, AWS STS checks to see if the originating identity pool is in the same AWS account as the IAM role. If AWS STS determines that the request is cross-account, it checks to see if the role trust policy has an `aud` condition. The assume-role call fails if no such conditions are present in the role trust policy. If the request is not cross-account, AWS STS doesn't enforce this restriction. As a best practice, always apply a condition of this type to the trust policies of your identity pool roles.

Additional trust policy conditions

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 that you attach to a role apply to all users who assume that role. To partition your users' access, use policy conditions and variables. For more information, see [IAM policy elements: Variables and tags](#). You can use the sub condition to restrict actions to Amazon Cognito identity IDs in your access policies. Use this option with caution, particularly for unauthenticated identities,

which lack a consistent user ID. For more information about the IAM policy variables for web federation with Amazon Cognito, see [IAM and AWS STS condition context keys](#) in the *AWS Identity and Access Management User Guide*.

For additional security protection, Amazon Cognito applies a scope-down policy to credentials that you assign your unauthenticated users in the [enhanced flow](#), using `GetCredentialsForIdentity`. The scope-down policy adds an [Inline session policy](#) and an [AWS managed session policy](#) to the IAM policies that you apply to your unauthenticated role. Because you must grant access in both the IAM policies for your role and the session policies, the scope-down policy limits users' access to services other than those in the list that follows.

Note

In the basic (classic) flow, you make your own [AssumeRoleWithWebIdentity](#) API request, and can apply these restrictions to the request. As a best security practice, don't assign any permissions above this scope-down policy to unauthenticated users.

Amazon Cognito also prevents authenticated and unauthenticated users from making API requests to Amazon Cognito identity pools and Amazon Cognito Sync. Other AWS services might place restrictions on service access from web identities.

In a successful request with the enhanced flow, Amazon Cognito makes an `AssumeRoleWithWebIdentity` API request in the background. Among the parameters in this request, Amazon Cognito includes the following.

1. Your user's identity ID.
2. The ARN of the IAM role that your user wants to assume.
3. A policy parameter that adds an *inline session policy*.
4. A `PolicyArns.member.N` parameter whose value is an *AWS managed policy* that grants additional permissions in Amazon CloudWatch.

Services that unauthenticated users can access

When you use the enhanced flow, the scope-down policies that Amazon Cognito applies to your user's session prevent them from using any services other than those listed in the following table. For a subset of services, only specific actions are allowed.

Category	Service
Analytics	Amazon Data Firehose
	Amazon Managed Service for Apache Flink
Application Integration	Amazon Simple Queue Service
AR & VR	Amazon Sumerian ¹
Business Applications	Amazon Mobile Analytics
	Amazon Simple Email Service
Compute	AWS Lambda
Cryptography & PKI	AWS Key Management Service ¹
Database	Amazon DynamoDB
	Amazon SimpleDB
Front-end Web & Mobile	AWS AppSync
	Amazon Location Service
	Amazon Simple Notification Service
	Amazon Pinpoint
	Amazon Location Service
Game Development	Amazon GameLift
Internet of Things (IoT)	AWS IoT
Machine Learning	Amazon CodeWhisperer
	Amazon Comprehend
	Amazon Lex
	Amazon Machine Learning

Category	Service
	Amazon Personalize Amazon Polly Amazon Rekognition Amazon SageMaker ¹ Amazon Textract ¹ Amazon Transcribe Amazon Translate
Management & Governance	Amazon CloudWatch Amazon CloudWatch Logs
Networking & Content Delivery	Amazon API Gateway
Security, Identity, & Compliance	Amazon Cognito user pools
Storage	Amazon Simple Storage Service

¹ For the AWS services in the following table, the inline policy grants a subset of actions. The table displays the available actions in each.

AWS service	Maximum permissions for unauthenticated enhanced flow users
AWS Key Management Service	Encrypt Decrypt ReEncryptTo ReEncryptFrom GenerateDataKey

AWS service	Maximum permissions for unauthenticated enhanced flow users
	GenerateDataKeyPair
	GenerateDataKeyPair
	GenerateDataKeyPairWithoutPlaintext
	GenerateDataKeyWithoutPlaintext
Amazon SageMaker	InvokeEndpoint
Amazon Textract	DetectDocumentText
	AnalyzeDocument
Amazon Sumerian	View*
Amazon Location Service	SearchPlaceIndex*
	GetPlace
	CalculateRoute*
	*Geofence
	*Geofences
	DevicePosition

To grant access to AWS services beyond this list, activate the **basic (classic) authentication flow** in your identity pool. If your users see NotAuthorizedException errors from AWS services that are allowed by the policies assigned to the IAM role for unauthenticated users, evaluate whether you can remove that service from your use case. If you can't, switch to the basic flow.

The inline session policy for guest users

Amazon Cognito first applies an inline policy in the request for IAM credentials. The inline session policy restricts your user's effective permissions from including access to any AWS services outside those in the following list. You must also grant permissions to these AWS services in the policies

that you apply to the user's IAM role. A user's effective permissions for an assumed-role session are the intersection of the policies assigned to their role, and their session policy. For more information, see [Session policies in the AWS Identity and Access Management User Guide](#).

Amazon Cognito adds the following inline policy to sessions for your users in AWS Regions that are enabled by default. For an overview of the net effect of the inline policy and other session policies, see [Services that unauthenticated users can access](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:*",  
                "logs:*",  
                "dynamodb:*",  
                "kinesis:*",  
                "mobileanalytics:*",  
                "s3:*",  
                "ses:*",  
                "sns:*",  
                "sqs:*",  
                "lambda:*",  
                "machinelearning:*",  
                "execute-api:*",  
                "iot:*",  
                "gamelift:*",  
                "scs:*",  
                "cognito-identity:*",  
                "cognito-idp:*",  
                "lex:*",  
                "polly:*",  
                "comprehend:*",  
                "translate:*",  
                "transcribe:*",  
                "rekognition:*",  
                "mobiletargeting:*",  
                "firehose:*",  
                "appsync:*",  
                "personalize:*",  
                "sagemaker:InvokeEndpoint",  
                "cognito-sync:*",  
            ]  
        }  
    ]  
}
```

```
        "sumerian:View*",
        "codewhisperer:*",
        "textract:DetectDocumentText",
        "textract:AnalyzeDocument",
        "sdb:/*"
    ],
    "Resource": [
        "*"
    ]
}
]
```

For all other Regions, the inline scope-down policy includes everything listed in the default Regions except for the following Action statements.

```
"cognito-sync:*",
"sumerian:View*",
"codewhisperer:*",
"textract:DetectDocumentText",
"textract:AnalyzeDocument",
"sdb:/*"
```

The AWS managed session policy for guests

Amazon Cognito also applies an AWS managed policy as a session policy to the enhanced-flow sessions of unauthenticated guests. This policy limits the scope of unauthenticated users' permissions with the policy `AmazonCognitoUnAuthedIdentitiesSessionPolicy`.

You must also grant this permission in the policies that you attach to your unauthenticated IAM role. A user's effective permissions for an assumed-role session are the intersection of the IAM policies assigned to their role, and their session policies. For more information, see [Session policies](#) in the *AWS Identity and Access Management User Guide*.

For an overview of the net effect of this AWS managed policy and other session policies, see [Services that unauthenticated users can access](#).

The `AmazonCognitoUnAuthedIdentitiesSessionPolicy` managed policy has the following permissions.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    "Effect": "Allow",
    "Action": [
        "rum:PutRumEvents",
        "polly:*",
        "comprehend:*",
        "translate:*",
        "transcribe:*",
        "rekognition:*",
        "mobiletargeting:*",
        "firehose:*",
        "personalize:*",
        "sagemaker:InvokeEndpoint",
        "geo:GetMap*",
        "geo:SearchPlaceIndex*",
        "geo:GetPlace",
        "geo:CalculateRoute*",
        "geo:*Geofence",
        "geo:*Geofences",
        "geo:*DevicePosition*",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncryptTo",
        "kms:ReEncryptFrom",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyPair",
        "kms:GenerateDataKeyPairWithoutPlaintext",
        "kms:GenerateDataKeyWithoutPlaintext"
    ],
    "Resource": "*"
]
}
```

Access policy examples

In this section, you can find example Amazon Cognito access policies that grant your users the minimum permissions necessary to do 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.

Grant an identity read access to a single object in Amazon 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:::amzn-s3-demo-bucket/assets/my_picture.jpg"]  
    }  
  ]  
}
```

Grant an identity both read and write access to identity specific paths in Amazon 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} can get, put, and list objects into arn:aws:s3:::amzn-s3-demo-bucket/us-east-1:12345678-1234-1234-1234-123456790ab. However, the identity would not be granted access to other objects in arn:aws:s3:::amzn-s3-demo-bucket.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "s3:PutObject",  
        "s3:DeleteObject",  
        "s3:ListBucket"  
      ],  
      "Effect": "Allow",  
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/us-east-1:12345678-1234-1234-1234-123456790ab/*"]  
    }  
  ]  
}
```

```
"Action": ["s3>ListBucket"],
"Effect": "Allow",
"Resource": ["arn:aws:s3:::amzn-s3-demo-bucket"],
"Condition": {"StringLike": {"s3:prefix": ["${cognito-identity.amazonaws.com:sub}/*"]}}}
},
{
  "Action": [
    "s3GetObject",
    "s3PutObject"
  ],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/${cognito-identity.amazonaws.com:sub}/*"]
}
]
```

Assign identities fine-grained access to Amazon DynamoDB

The following access policy provides fine-grained access control to DynamoDB resources using Amazon Cognito environment variables. These variables 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}"]
        }
    }
}
]
```

Grant an identity permission to invoke a Lambda function

The following access policy grants an identity permission to invoke a Lambda function.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
            ]
        }
    ]
}
```

Grant an identity permission to publish records to Kinesis Data Streams

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"
            ]
        }
    ]
}
```

```
]  
}
```

Grant an identity access to their data in the Amazon Cognito Sync store

The following access policy grants an identity permissions to access only their own 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. The following is 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 grants federated users from `cognito-identity.amazonaws.com` (the issuer of the OpenID Connect token) permission 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 one of the array members of the multi-value `amr` claim of the token issued by the Amazon Cognito GetOpenIdToken API operation has the value `unauthenticated`.

When Amazon Cognito creates a token, it sets the `amr` of the token as either `unauthenticated` or `authenticated`. If `amr` is `authenticated`, the token includes any providers used during authentication. This means that you can create a role that trusts only users that logged in via Facebook by changing the `amr` condition as shown:

```
"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 you don't configure your role correctly to trust your identity pool, an exception from STS results, like the following:

```
AccessDenied -- Not authorized to perform sts:AssumeRoleWithWebIdentity
```

If you see this message, check that your identity pool and authentication type have an appropriate role.

Security best practices for Amazon Cognito identity pools

Amazon Cognito identity pools provide temporary AWS credentials for your application. AWS accounts often contain both the resources that your application users need, and private back-end resources. The IAM roles and policies that make up AWS credentials can grant access to any of these resources.

The primary best practice of identity pool configuration is to ensure that your application can get the job done without excess or unintended privilege. To guard against security misconfiguration, review these recommendations before the launch of each application that you want to release to production.

Topics

- [IAM configuration best practices](#)
- [Identity pool configuration best practices](#)

IAM configuration best practices

When a guest or authenticated user initiates a session in your application that requires identity pool credentials, your application retrieves temporary AWS credentials for an IAM role. The credentials might be for a default role, a role chosen by rules in your identity pool configuration, or for a custom role chosen by your app. With the permissions assigned to each role, your user gains access to your AWS resources.

For more information about general IAM best practices, see [IAM best practices](#) in the AWS Identity and Access Management User Guide.

Use trust policy conditions in IAM roles

IAM requires that roles for identity pools have at least one trust policy condition. This condition can, for example, set the role's scope to authenticated users only. AWS STS also requires that cross-account basic authentication requests have two specific conditions: `cognito-identity.amazonaws.com:aud` and `cognito-identity.amazonaws.com:amr`. As a best practice, apply both of these conditions in all IAM roles that trust the identity pools service principal `cognito-identity.amazonaws.com`.

- `cognito-identity.amazonaws.com:aud`: The `aud` claim in the identity pool token must match a trusted identity pool ID.
- `cognito-identity.amazonaws.com:amr`: The `amr` claim in the identity pool token must be either *authenticated* or *unauthenticated*. With this condition, you can reserve access to a role only to unauthenticated guests, or only to authenticated users. You can further refine the value of this condition to restrict the role to users from a specific provider, for example `graph.facebook.com`.

The following example role trust policy grants access to a role under the following conditions:

```
{  
  "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:a1b2c3d4-5678-90ab-cdef-  
EXAMPLE11111"  
        },  
        "ForAnyValue:StringLike": {  
            "cognito-identity.amazonaws.com:amr": "authenticated"  
        }  
    }  
}  
]  
}
```

Elements that relate to identity pools

- "Federated": "cognito-identity.amazonaws.com": Users must come from an identity pool.
- "cognito-identity.amazonaws.com:aud": "us-east-1:a1b2c3d4-5678-90ab-cdef-example11111": Users must come from the specific identity pool us-east-1:a1b2c3d4-5678-90ab-cdef-example11111.
- "cognito-identity.amazonaws.com:amr": "authenticated": Users must be authenticated. Guest users can't assume the role.

Apply least privilege permissions

When you set permissions with IAM policies for authenticated access or guest access, grant only the specific permissions required to perform specific tasks, or *least privilege* permissions. The following example IAM policy, when applied to a role, grants read-only access to a single image file in an Amazon S3 bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
{  
    "Action": [  
        "s3:GetObject"  
    ],  
    "Effect": "Allow",  
    "Resource": ["arn:aws:s3:::mybucket/assets/my_picture.jpg"]  
}  
]  
}
```

Identity pool configuration best practices

Identity pools have flexible options for the generation of AWS credentials. Don't take design shortcuts when your application can work with the most secure methods.

Understand the effects of guest access

Unauthenticated guest access permits users to retrieve data from your AWS account before they sign in. Anyone who knows your identity pool ID can request unauthenticated credentials. Your identity pool ID isn't confidential information. When you activate guest access, the AWS permissions that you grant to unauthenticated sessions are available to everyone.

As a best practice, leave guest access deactivated and fetch required resources only after users authenticate. If your application requires access to resources before sign-in, take the following precautions.

- Familiarize yourself with the [automatic limitations placed on unauthenticated roles](#).
- Monitor and adjust the permissions of your unauthenticated IAM roles to match the specific needs of your application.
- Grant access to specific resources.
- Secure the trust policy of your default unauthenticated IAM role.
- Activate guest access only when you are confident that you would grant the permissions in your IAM role to anyone on the internet.

Use enhanced authentication by default

With basic (classic) authentication, Amazon Cognito delegates selection of the IAM role to your app. In contrast, the enhanced flow uses the centralized logic in your identity pool to determine the IAM role. It also provides additional security for unauthenticated identities with a [scope-down](#)

[policy](#) that sets an upper limit on IAM permissions. The enhanced flow is the most secure choice with the lowest level of developer effort. To learn more about these options, see [Identity pools authentication flow](#).

The basic flow can expose the client-side logic that goes into role selection and assembly of the AWS STS API request for credentials. The enhanced flow hides both the logic and the assume-role request behind identity pool automation.

When you configure basic authentication, apply [IAM best practices](#) to your IAM roles and their permissions.

Use developer providers securely

Developer authenticated identities are a feature of identity pools for server-side applications. The only evidence of authentication that identity pools require for developer authentication are the AWS credentials of an identity pool developer. Identity pools don't enforce any restrictions on the validity of the developer-provider identifiers that you present in this authentication flow.

As a best practice, only implement developer providers under the following conditions:

- To create accountability for the use of developer-authenticated credentials, design your developer provider name and identifiers to indicate the authentication source. For example: "Logins" : {"MyCorp provider" : "[*provider application ID*]"}.
- Avoid long-lived user credentials. Configure your server-side client to request identities with service-linked roles like [EC2 instance profiles](#) and [Lambda execution roles](#).
- Avoid mixing internal and external sources of trust in the same identity pool. Add your developer provider and your single sign-on (SSO) providers in separate identity pools.

Using attributes for access control

Attributes for access control is the Amazon Cognito identity pools implementation of attribute-based access control (ABAC). 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 a 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 the 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 for access control with Amazon Cognito identity pools](#)
- [Using attributes for access control policy example](#)
- [Turn off attributes for access control \(console\)](#)
- [Default provider mappings](#)

Using attributes to control access has several benefits:

- Permissions management is more efficient 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](#)
- [Set up an SDK](#)
- [Integrated identity providers](#)
- [Credentials](#)

To use attributes for access control, the **Claim** that you set as the source of data sets the value of the **Tag Key** that you choose. Amazon Cognito applies the tag key and value to your user's session. Your IAM policies can evaluate your user's access from the \${aws:PrincipalTag/*tagkey*} condition. IAM evaluates the value of your user's tag against the policy.

You must prepare IAM roles whose credentials you want to pass to your users. The trust policy of these roles must permit Amazon Cognito to assume the role for your user. For attributes for access control, you must also allow Amazon Cognito to apply principal tags to your user's temporary session. Grant permission to assume the role with the action [AssumeRoleWithWebIdentity](#). Grant permission to tag users' sessions with the [permission-only action](#) `sts:TagSession`. For more information, see [Passing session tags in AWS Security Token Service](#) in the *AWS Identity and Access Management User Guide*. For an example trust policy that grants `sts:AssumeRoleWithWebIdentity` and `sts:TagSession` permissions to the Amazon Cognito service principal `cognito-identity.amazonaws.com`, see [Using attributes for access control policy example](#).

To configure attributes for access control in the console

1. Sign in to the [Amazon Cognito console](#) and select **Identity pools**. Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Identity providers**. Choose the identity provider that you want to edit. If you want to add a new IdP, select **Add identity provider**.
4. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, choose **Edit** in **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

5. Select **Save changes**.

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 makes sure 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"
        }
    }
}
]
```

Turn off attributes for access control (console)

Follow this procedure to deactivate attributes for access control.

To deactivate attributes for access control in the console

1. Sign in to the [Amazon Cognito console](#) and select **Identity pools**. Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Identity providers**. Choose the identity provider that you want to edit.
4. Choose **Edit** in **Attributes for access control**.
5. To apply no principal tags, choose **Inactive**.
6. Select **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-98b8-45a6729b4c88"
Facebook	Access token	aud(app_id), sub(user_id)	"492844718097981", "112177216992379"

Provider	Token type	Principal tag values	Example
Google	ID token	aud(client ID) and sub(user ID)	"620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.apps.googleusercontent.com", "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.compute-1.client", "001968.a6ca34e9c1e742458a26cf8005854be9.0733"
Amazon	Access token	aud (Client ID on Amzn Dev Ac), user_id(user ID)	"amzn1.application-oa2-client.9d70d9382d3446108aaee3dd763a0fa6" , "amzn1.acount.AGHNIFJQMFSBG3G6XCPVB35ORQAA"

Provider	Token type	Principal tag values	Example
Standard OIDC providers	ID and access tokens	aud (as client_id), sub (as user ID)	"620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.apps.googleusercontent.com", "109220063452404746097"
Twitter	Access token	aud (app ID; app Secret), sub (user ID)	"DfwifTtKEX1FilBRnOTlR0CFK;Xgj5xb8xlrIVCPjXgLIdkW7fXmwkJJrFvnoK9gwZkLexo1y5z1", "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.

Using 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 one of the array members of the multi-value `amr` claim of the token issued by the Amazon Cognito `GetOpenIdToken` API action has the value `authenticated`.

Granting pass-role permission

To allow a user to set roles with permissions in excess of the user's existing permissions on an identity pool, grant them `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 Amazon Resource Name (ARN). You must also attach this policy to your user. 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 [Working with user attributes](#).

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 request**.

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.

To add rule-based mapping to an identity provider in the Amazon Cognito console, add or update an IdP and select **Choose role with rules** under **Role selection**. From there, you can add rules that map provider claims to IAM roles.

You can set up rule-based mapping for identity providers in the AWS CLI or API with the `RulesConfiguration` field of the [RoleMapping](#) type. You can specify this field in the `RoleMappings` parameter of the [SetIdentityPoolRoles](#) API.

For example, the following AWS CLI command adds a rule that assigns the role `arn:aws:iam::123456789012:role/Sacramento_team_S3_admin` to users in your Sacramento location who were authenticated by OIDC IdP `arn:aws:iam::123456789012:oidc-provider/myOIDCIdP`:

```
aws cognito-identity set-identity-pool-roles --region us-east-1 --cli-input-json file://role-mapping.json
```

Contents of `role-mapping.json`:

```
{
  "IdentityPoolId": "us-east-1:12345678-corner-cafe-123456790ab",
  "Roles": {
    "authenticated": "arn:aws:iam::123456789012:role/myS3WriteAccessRole",
    "unauthenticated": "arn:aws:iam::123456789012:role/myS3ReadAccessRole"
  },
  "RoleMappings": {
    "arn:aws:iam::123456789012:oidc-provider/myOIDCIdP": {
```

```
"Type": "Rules",
"AmbiguousRoleResolution": "AuthenticatedRole",
"RulesConfiguration": {
    "Rules": [
        {
            "Claim": "locale",
            "MatchType": "Equals",
            "Value": "Sacramento",
            "RoleARN": "arn:aws:iam::123456789012:role/
Sacramento_team_S3_admin"
        }
    ]
}
}
```

For each user pool or other authentication provider that you configure for an identity pool, you can create up to 25 rules. This limit is not adjustable. For more information, see [Quotas in Amazon Cognito](#).

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 must reassociate your roles with your identity pool 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.

Web identity credentials providers are part of the default credential provider chain in AWS SDKs. To set your identity pool token in a local config file for an AWS SDK or the AWS CLI, add a `web_identity_token_file` profile entry. See [Assume role credential provider](#) in the AWS SDKs and Tools Reference Guide.

To learn more about how to populate web identity credentials in your SDK, refer to the SDK developer guide. For best results, start your project with the identity pool integration that's built in to AWS Amplify.

AWS SDK resources for getting and setting credentials with identity pools

- [Identity Pool Federation](#) (Android) in the Amplify Dev Center
- [Identity Pool Federation](#) (iOS) in the Amplify Dev Center
- [Using Amazon Cognito Identity to authenticate users](#) in the AWS SDK for JavaScript Developer Guide
- [Amazon Cognito credentials provider](#) in the AWS SDK for .NET Developer Guide
- [Specify Credentials Programmatically](#) in the AWS SDK for Go Developer Guide
- [Supply temporary credentials in code](#) in the AWS SDK for Java 2.x Developer Guide
- [assumeRoleWithWebIdentityCredentialProvider](#) provider in the AWS SDK for PHP Developer Guide
- [Assume Role With Web Identity Provider](#) in the AWS SDK for Python (Boto3) documentation
- [Specifying your credentials and default region](#) in the AWS SDK for Rust Developer Guide

The following sections provide example code in some legacy AWS SDKs.

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.

To use a Amazon Cognito identity pool in an Android app, set up AWS Amplify. For more information, see [Authentication](#) in the *Amplify Dev Center*.

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 must 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, complete the following steps.

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see [Swift Authentication](#) and [Flutter Authentication](#) in the *Amplify Dev Center*.

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 [Amplify iOS SDK 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.

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see [Swift Authentication](#) in the *Amplify Dev Center*.

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 [Amplify iOS SDK 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.

The [AWS SDK for Unity](#) is now part of the [AWS SDK for .NET](#). To get started with Amazon Cognito in the AWS SDK for .NET, see [Amazon Cognito credentials provider](#) in the AWS SDK for .NET Developer Guide. Or see [Amplify Dev Center](#) for options for building an app with AWS Amplify.

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.

The [AWS SDK for Xamarin](#) is now part of the [AWS SDK for .NET](#). To get started with Amazon Cognito in the AWS SDK for .NET, see [Amazon Cognito credentials provider](#) in the AWS SDK for .NET Developer Guide. Or see [Amplify Dev Center](#) for options for building an app with AWS Amplify.

Note

Note: If you created your identity pool before February 2015, you must 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 with temporary credentials

The result of a successful authentication with an identity pool is a set of AWS credentials. With these credentials, your application can make requests to AWS resources that are protected with IAM authentication. With the various AWS SDKs that you can add to your applications to access identity pools API operations, you can make unauthenticated API requests that produce temporary credentials. Then you can add SDKs for other AWS services to your client and sign requests with those temporary credentials. The IAM permissions granted to your temporary-credentials role must permit the operations that you request from other services.

After you configure your Amazon Cognito credentials provider and retrieve AWS credentials, create an AWS service client. The following are some examples from AWS SDK documentation.

AWS SDK resources for creating a client

- [AWS Client configuration](#) in the AWS SDK for C++ Developer Guide
- [Using the AWS SDK for Go V2 with AWS services](#) in the AWS SDK for Go Developer Guide
- [Configuring HTTP clients](#) in the AWS SDK for Java 2.x Developer Guide
- [Creating and calling service objects](#) in the AWS SDK for JavaScript Developer Guide
- [Creating clients](#) in the AWS SDK for Python (Boto3) documentation
- [Creating a service client](#) in the AWS SDK for Rust Developer Guide
- [Using clients](#) in the AWS SDK for Swift Developer Guide

The following snippet initializes an Amazon DynamoDB client:

Android

To use a Amazon Cognito identity pool in an Android app, set up AWS Amplify. For more information, see [Authentication](#) in the *Amplify Dev Center*.

```
// 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

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see [Swift Authentication](#) and [Flutter Authentication](#) in the *Amplify Dev Center*.

```
// 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

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see [Swift Authentication](#) in the *Amplify Dev Center*.

```
// 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

The [AWS SDK for Unity](#) is now part of the [AWS SDK for .NET](#). To get started with Amazon Cognito in the AWS SDK for .NET, see [Amazon Cognito credentials provider](#) in the AWS SDK for .NET Developer Guide. Or see [Amplify Dev Center](#) for options for building an app with AWS Amplify.

```
// 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

The [AWS SDK for Xamarin](#) is now part of the [AWS SDK for .NET](#). To get started with Amazon Cognito in the AWS SDK for .NET, see [Amazon Cognito credentials provider](#) in the AWS SDK for .NET Developer Guide. Or see [Amplify Dev Center](#) for options for building an app with AWS Amplify.

```
// 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 third-party identity providers

With Amazon Cognito identity pools, you can integrate with a variety of external identity providers (IdPs) to provide temporary AWS credentials through federated authentication in your application. By configuring your identity pool to work with these external IdPs, you can authorize access to back-end AWS resources for your users with authentication by Amazon Cognito user pools, social providers, OIDC providers, or SAML providers. This section covers the steps to set up and integrate IdPs with your Amazon Cognito identity pool.

Using the `logins` property, you can set credentials received from an identity provider (IdP). You can also associate an identity pool with multiple IdPs. For example, you can set both the Facebook and Google tokens in the `logins` property to associate the unique Amazon Cognito identity with both IdP logins. The user can authenticate with either account, but Amazon Cognito returns the same user identifier.

The following instructions guide you through authentication with the IdPs that Amazon Cognito identity pools support.

Topics

- [Setting up Facebook as an identity pools IdP](#)
- [Setting up Login with Amazon as an identity pools IdP](#)
- [Setting up Google as an identity pool IdP](#)
- [Setting up Sign in with Apple as an identity pool IdP](#)
- [Setting up an OIDC provider as an identity pool IdP](#)
- [Setting up a SAML provider as an identity pool IdP](#)

Setting up Facebook as an identity pools IdP

Amazon Cognito identity pools work with Facebook to provide federated authentication for your application users. This section explains how to register and set up your application with Facebook as an IdP.

Set up Facebook

Register your application with Facebook before you authenticate Facebook users and interact with Facebook APIs.

The [Facebook Developers portal](#) helps you to set up your application. Do this procedure before you integrate Facebook in your Amazon Cognito identity pool:

 **Note**

Amazon Cognito identity pools federation isn't compatible with [Facebook Limited Login](#).

For more information about how to set up Facebook Login for iOS without exceeding the permissions set for Limited Login, see [Facebook Login for iOS - Quickstart](#) at *Meta for Developers*.

Setting 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

For more information about how to integrate Android apps with Facebook Login, see the [Facebook Getting Started Guide](#).

iOS - Objective-C

For more information about how to integrate iOS Objective-C apps with Facebook Login, see the [Facebook Getting Started Guide](#).

iOS - Swift

For more information about how to integrate iOS Swift apps with Facebook Login, see the [Facebook Getting Started Guide](#).

JavaScript

For more information about how to integrate JavaScript web apps with Facebook Login, see the [Facebook Getting Started Guide](#).

Unity

For more information about how to integrate Unity apps with Facebook Login, see the [Facebook Getting Started Guide](#).

Xamarin

To add Facebook authentication, first follow the appropriate flow below to integrate the Facebook SDK into 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 an identity provider in the Amazon Cognito identity pools console

Use the following procedure to configure your identity provider.

To add a Facebook identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Facebook**.
5. Enter the **App ID** of the OAuth project that you created at [Meta for Developers](#). For more information, see [Facebook Login](#) in the *Meta for Developers Docs*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.

- b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

Using Facebook

Android

To add Facebook authentication, first follow the [Facebook guide](#) and integrate the Facebook SDK into 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.

After you authenticate 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 that matches this particular Facebook identity. If the user already exists, the API returns the existing identifier. Otherwise, the API returns a new identifier. The client SDK automatically caches identifiers on the local device.

Note

After you set the logins map, make a call to `refresh` or `get` to retrieve the AWS credentials.

iOS - Objective-C

To add Facebook authentication, first follow the [Facebook guide](#) and 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.

When you implement the `logins` method, return a dictionary that contains `AWSIdentityProviderFacebook`. This dictionary acts as the key, and the current access token from the authenticated Facebook user acts 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](#) and 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).

 **Note**

Amazon Cognito identity pools federation isn't compatible with [Facebook Limited Login](#).

For more information about how to set up Facebook Login for iOS without exceeding the permissions set for Limited Login, see [Facebook Login for iOS - Quickstart](#) at *Meta for Developers*.

To provide the Facebook access token to Amazon Cognito, implement the [AWSIdentityProviderManager](#) protocol.

When you implement the `logins` method, return a dictionary containing `AWSIdentityProviderFacebook`. This dictionary acts as the key, and the current access token from the authenticated Facebook user acts 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 add Facebook authentication, follow the [Facebook Login for the Web](#) and 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.

After you authenticate 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 you set the logins map, 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 add Facebook authentication, first follow the [Facebook guide](#) and integrate the Facebook SDK into 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.

After you authenticate 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);
}
```

Before you use FB.AccessToken, call FB.Login() and make sure FB.IsLoggedIn is true.

Xamarin

Xamarin for Android:

```
public void InitializeFacebook() {
    FacebookSdk.SdkInitialize(this.ApplicationContext);
    callbackManager = CallbackManagerFactory.Create();
    LoginManager.Instance.RegisterCallback(callbackManager, new FacebookCallback <LoginResult> () {
        HandleSuccess = loginResult => {
            var accessToken = loginResult.AccessToken;
            credentials.AddLogin("graph.facebook.com", accessToken.Token);
            //open new activity
        },
        HandleCancel = () => {
            //throw error message
        },
        HandleError = loginError => {
            //throw error message
        }
    });
    LoginManager.Instance.LogInWithReadPermissions(this, new List <string> {
        "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
    }
});
}
```

Setting up Login with Amazon as an identity pools IdP

Amazon Cognito identity pools work with Login with Amazon to provide federated authentication for your mobile and web app users. This section explains how to register and set up your application with Login with Amazon as an identity provider (IdP).

Set up Login with Amazon to work with Amazon Cognito in the [Developer Portal](#). For more information, see [Setting Up Login with Amazon](#) in the Login with Amazon FAQ.

 **Note**

To integrate Login with Amazon into a Xamarin application, follow the [Xamarin Getting Started Guide](#).

 **Note**

You can't natively integrate Login with Amazon on the Unity platform. Instead, use a web view and go through the browser sign-in flow.

Setting up Login with Amazon

Implement Login with Amazon

In the [Amazon developer portal](#), you can set up an OAuth application to integrate with your identity pool, find Login with Amazon documentation, and download SDKs. Choose **Developer console**, then **Login with Amazon** in the developer portal. You can create a security profile for your application and then build Login with Amazon authentication mechanisms into your app. See [Getting credentials](#) for more information about how to integrate Login with Amazon authentication with your app.

Amazon issues an OAuth 2.0 **client ID** for your new security profile. You can find the **client ID** on the security profile **Web Settings** tab. Enter the **Security Profile ID** in the **App ID** field of the Login with Amazon IdP in your identity pool.

 **Note**

You enter the **Security Profile ID** in the **App ID** field of the Login with Amazon IdP in your identity pool. This differs from user pools, which use **client ID**.

Configure the external provider in the Amazon Cognito console

To add a Login with Amazon identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Login with Amazon**.
5. Enter the **App ID** of the OAuth project that you created at [Login with Amazon](#). For more information, see [Login with Amazon Documentation](#).
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.

- b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

Use Login with Amazon: Android

After you authenticate 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

After you authenticate 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 = @{@"AWSLoginProviderKeyLoginWithAmazon":  
        apiResult.result};  
    }  
}
```

Use Login with Amazon: iOS - Swift

After you authenticate 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);
    btnLoginSetTitle ("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());
}

}
```

Setting up Google as an identity pool IdP

Amazon Cognito identity pools work 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 IdP.

Android

Note

If your app uses Google and is available on multiple mobile platforms, you should configure it as an [OpenID Connect Provider](#). Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see [Cross-client Identity](#).

Setting up Google

To activate Google Sign-in for Android, create a Google Developers console project for your application.

1. Go to the [Google Developers console](#) and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **Android** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and then choose **Create and continue**.

5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see [Creating and managing projects](#) in the Google Cloud documentation.

For more information about how to integrate Google into your Android app, see [Authenticate users with Sign in with Google](#) in the Google Identity documentation.

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at [Google Cloud Platform](#). For more information, see [Setting up OAuth 2.0](#) in *Google Cloud Platform Console Help*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.

- b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

Use Google

To enable login with Google in your application, follow the instructions in the [Google documentation for Android](#). When a user signs in, they request an OpenID Connect authentication token from Google. Amazon Cognito then uses the token to authenticate the user and generate a unique identifier.

The following example 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 is available on multiple mobile platforms, configure Google as an [OpenID Connect Provider](#). Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see [Cross-client Identity](#).

Setting up Google

To enable Google Sign-in for iOS, create a Google Developers console project for your application.

1. Go to the [Google Developers console](#) and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **iOS** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account. Choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see [Creating and managing projects](#) in the Google Cloud documentation.

For more information about how to integrate Google into your iOS app, see [Google Sign-In for iOS](#) in the Google Identity documentation.

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at [Google Cloud Platform](#). For more information, see [Setting up OAuth 2.0 in Google Cloud Platform Console Help](#).
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the

- Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
- ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
- a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **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 is available on multiple mobile platforms, configure Google as an [OpenID Connect Provider](#). Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see [Cross-client Identity](#).

Setting up Google

To enable Google Sign-in for iOS, create a Google Developers console project for your application.

1. Go to the [Google Developers console](#) and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **iOS** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see [Creating and managing projects](#) in the Google Cloud documentation.

For more information about how to integrate Google into your iOS app, see [Google Sign-In for iOS](#) in the Google Identity documentation.

Choose **Manage Identity Pools** from the [Amazon Cognito Console home page](#):

Configuring the external provider in the Amazon Cognito Console

1. Choose the name of the identity pool where 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 the section.
4. Choose the **Google** tab.
5. Choose **Unlock**.
6. Enter the Google Client ID that 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 that Amazon Cognito uses to authenticate the user and generate a unique identifier.

Successful authentication results in a GTMOAuth2Authentication object that contains an id_token. Amazon Cognito uses this token 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 is available on multiple mobile platforms, you should configure Google as an [OpenID Connect Provider](#). Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see [Cross-client Identity](#).

Setting up Google

To enable Google Sign-in for a JavaScript web app, create a Google Developers console project for your application.

1. Go to the [Google Developers console](#) and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks their consent to share their profile data with your app.

3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **Web application** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see [Creating and managing projects](#) in the Google Cloud documentation.

For more information about how to integrate Google into your web app, see [Sign in With Google](#) in the Google Identity documentation.

Configure the External Provider in the Amazon Cognito Console

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at [Google Cloud Platform](#). For more information, see [Setting up OAuth 2.0](#) in *Google Cloud Platform Console Help*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

- ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
- a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **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` that 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

Setting up Google

To enable Google Sign-in for a Unity app, create a Google Developers console project for your application.

1. Go to the [Google Developers console](#) and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **Web application** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. For Unity, create an additional **OAuth client ID** for **Android**, and another for **iOS**.
5. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
6. Grant the service account access to your project. Grant users access to the service account as your app requires.
7. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see [Creating and managing projects](#) in the Google Cloud documentation.

Create an OpenID Provider in the IAM Console

1. Create an OpenID Provider in the IAM Console. For information about how to set up an OpenID Provider, see [Using OpenID Connect Identity Providers](#).
2. When prompted for your Provider URL, enter "https://accounts.google.com".
3. When prompted to enter a value in the **Audience** field, enter any one of the three client IDs that you created in the previous steps.
4. Choose the provider name and add two more audiences with the two other client IDs.

Configure the External Provider in the Amazon Cognito Console

Choose **Manage Identity Pools** from the [Amazon Cognito Console home page](#):

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at [Google Cloud Platform](#). For more information, see [Setting up OAuth 2.0](#) in *Google Cloud Platform Console Help*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **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, use the three IDs for the Android and iOS platforms to configure the plugin.

Use Google

The following example 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

Amazon Cognito doesn't natively support Google 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, 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 use an alternative client or the web flow in a web view.

After you have the token, you can set it in your `CognitoAWSCredentials`:

```
credentials.AddLogin("accounts.google.com", token);
```

 **Note**

If your app uses Google and is available on multiple mobile platforms, you should configure Google as an [OpenID Connect Provider](#). Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see [Cross-client Identity](#).

Setting up Sign in with Apple as an identity pool IdP

Amazon Cognito identity pools work 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 (IdP).

To add Sign in with Apple as an authentication provider to an identity pool, you must complete two procedures. First, integrate Sign in with Apple in an application, and then configure Sign in with Apple in identity pools. For the most up-to-date information about setting up Sign in with Apple, see [Configuring Your Environment for Sign in with Apple](#) in the Apple Developer documentation.

Set up Sign in with Apple

To configure Sign in with Apple as an IdP, 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 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 your 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, choose the **App IDs** menu, then **Services IDs**. Choose the + 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 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 URLs**, choose the + icon. For **Domains and Subdomains**, enter the domain name of your app. In **Return URLs**, enter the callback URL where the authorization redirects the user after they authenticate through Sign in with Apple.
 - 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.

To add a Sign in with Apple identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Sign in with Apple**.
5. Enter the **Services ID** of the OAuth project you created with [Apple Developer](#). For more information, see [Authenticating users with Sign in with Apple](#) in *Sign in with Apple Documentation*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the

- Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
- ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
- a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **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 IdP.

```
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 with Sign in with Apple, follow [Authenticating Users with 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: (ASAAuthorizationAppleIDCredential *)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 about how to set 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 through 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 through 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.

After you have the token, you can set it in your CognitoAWSCredentials:

```
credentials.AddLogin("appleid.apple.com", token);
```

Setting up an OIDC provider as an identity pool IdP

[OpenID Connect](#) is an open standard for authentication that a number of login providers support. With Amazon Cognito, you can link identities with OpenID Connect providers that you configure through [AWS Identity and Access Management](#).

Adding an OpenID Connect provider

For information about how to create an OpenID Connect provider, see [Creating OpenID Connect \(OIDC\) identity providers](#) in the *AWS Identity and Access Management User Guide*.

Associating a provider with Amazon Cognito

To add an OIDC identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **OpenID Connect (OIDC)**.
5. Choose an **OIDC identity provider** from the IAM IdPs in your AWS account. If you want to add a new SAML provider, choose **Create new provider** to navigate to the IAM console.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.

- b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

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.

After you have a token, add the token to the logins map. Use the URI of your provider as the key.

Validating an OpenID Connect token

When you first integrate with Amazon Cognito, you might receive an `InvalidToken` exception. It is important to understand how Amazon Cognito validates OpenID Connect (OIDC) tokens.

Note

As specified here (<https://tools.ietf.org/html/rfc7523>), Amazon Cognito provides a grace period of 5 minutes to handle any clock skew between systems.

1. The `iss` parameter must match the key that the logins map uses (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 public key matches the fingerprint that you set in IAM when you created your OIDC provider.
4. If the `azp` parameter is present, check this value against listed client IDs in your OIDC provider.
5. If the `azp` parameter isn't present, check the `aud` parameter against listed client IDs in your OIDC provider.

The website jwt.io is a valuable resource that you can use to decode tokens and 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.

When you implement the `logins` method, return a dictionary that contains the OIDC provider name that you configured. This dictionary acts as the key, and the current ID token from the authenticated user acts 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);
```

Setting up a SAML provider as an identity pool IdP

With Amazon Cognito identity pools, you can authenticate users with identity providers (IdPs) through SAML 2.0. You can use an IdP that supports SAML with Amazon Cognito to provide a simple onboarding flow for your users. Your SAML-supporting IdP specifies the IAM roles that your users can assume. This way, different users can receive different sets of permissions.

Configuring your identity pool for a SAML IdP

The following steps describe how to configure your identity pool to use a SAML-based IdP.

Note

Before you configure your identity pool to support a SAML provider, first configure the SAML IdP in the [IAM console](#). For more information, see [Integrating third-party SAML solution providers with AWS](#) in the *IAM User Guide*.

To add a SAML identity provider (IdP)

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **SAML**.
5. Choose a **SAML identity provider** from the IAM IdPs in your AWS account. If you want to add a new SAML provider, choose **Create new provider** to navigate to the IAM console.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
 - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
 - i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
 - ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
 - a. To apply no principal tags, choose **Inactive**.
 - b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.
 - c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

8. Select **Save changes**.

Configuring your SAML IdP

After you create the SAML provider, configure your SAML IdP to add relying party trust between your IdP and AWS. With many IdPs, you can specify a URL that the IdP can use to read relying party information and certificates from an XML document. 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 IdP to populate the claims that AWS needs. For details on the claim configuration, see [Configuring SAML assertions for authentication response](#).

When your SAML IdP includes more than one signing certificate in SAML metadata, at sign-in your user pool determines that the SAML assertion is valid if it matches any certificate in the SAML metadata.

Customizing your user role with SAML

When you use SAML with Amazon Cognito Identity, you can customize the role for the end user. Amazon Cognito only supports the [enhanced flow](#) with the SAML-based IdP. You don't need to specify an authenticated or unauthenticated role for the identity pool to use a SAML-based IdP. 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 can assume. You can configure the SAML IdP to populate the role attributes based on the user attribute information available from the IdP. If you receive multiple roles in the SAML assertion, populate the optional `customRoleArn` parameter when you call `getCredentialsForIdentity`. The user assumes this `customRoleArn` if the role matches one in the claim in the SAML assertion.

Authenticating users with a SAML IdP

To federate with the SAML-based IdP, determine the URL where the user initiates 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 IdP in Amazon Cognito, first authenticate users with your SAML identity provider from your iOS or Android application. The code that you use to integrate and authenticate with the SAML IdP is specific to SAML providers. After you authenticate your user, you can use Amazon Cognito APIs to provide the resulting SAML assertion to Amazon Cognito Identity .

You can't repeat, or *replay*, a SAML assertion in the Logins map of your identity pool API request. A replayed SAML assertion has an assertion ID that duplicates the ID of an earlier API request. API operations that can accept a SAML assertion in the Logins map include [GetId](#), [GetCredentialsForIdentity](#), [GetOpenIdToken](#), and [GetOpenIDTokenForDeveloperIdentity](#). You can replay a SAML assertion ID one time per API request in an identity pool authentication flow. For example, you can supply the same SAML assertion in a GetId request and a subsequent GetCredentialsForIdentity request, but not in a second GetId request.

Android

If you use 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

Amazon Cognito supports developer-authenticated identities, in addition to web identity federation through [Setting up Facebook as an identity pools IdP](#), [Setting up Google as an identity pool IdP](#), [Setting up Login with Amazon as an identity pools IdP](#), and [Setting up Sign in with Apple as an identity pool IdP](#). With developer-authenticated identities, you can register and authenticate users through 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, see [Understanding Amazon Cognito Authentication Part 2: Developer Authenticated Identities](#) in the AWS blog.

Understanding the authentication flow

The [GetOpenIdTokenForDeveloperIdentity](#) API operation can initiate developer authentication for both enhanced and basic authentication. This API authenticates a request with administrative credentials. The Logins map is an identity pool developer provider name like `login.mydevprovider` paired with a custom identifier.

Example:

```
"Logins": {  
    "login.mydevprovider": "my developer identifier"  
}
```

Enhanced authentication

Call the [GetCredentialsForIdentity](#) API operation with a Logins map with the name `cognito-identity.amazonaws.com` and a value of the token from `GetOpenIdTokenForDeveloperIdentity`.

Example:

```
"Logins": {
```

```
        "cognito-identity.amazonaws.com": "eyJra12345EXAMPLE"  
    }
```

GetCredentialsForIdentity with developer-authenticated identities returns temporary credentials for the default authenticated role of the identity pool.

Basic authentication

Call the [AssumeRoleWithWebIdentity](#) API operation and request the RoleArn of any IAM role that has an appropriate [trust relationship defined](#). Set the value of WebIdentityToken to the token obtained from GetOpenIdTokenForDeveloperIdentity.

For information on the developer-authenticated identities authflow and how they differ from external-provider identities, see [Identity pools authentication flow](#).

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:

To add a custom developer provider

1. Choose **Identity pools** from the [Amazon Cognito console](#). Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Custom developer provider**.
5. Enter a **Developer provider name**. You can't change or delete your developer provider after you add it.
6. Select **Save changes**.

Note: Once the provider name has been set, it cannot be changed.

Implement an identity provider

Android

To use developer-authenticated identities, implement your own identity provider class that extends `AWSAbstractCognitoIdentityProvider`. Your identity provider class should return a response object containing the token as an attribute.

Following is a basic 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.USWEST1);  
CognitoCachingCredentialsProvider credentialsProvider = new  
    CognitoCachingCredentialsProvider( context, developerProvider, Regions.USWEST1);
```

iOS - objective-C

To use developer-authenticated identities, implement your own identity provider class that extends [AWS Cognito Credentials Provider Helper](#). 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 that extends [AWSIdentityProviderHelper](#). 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 : AWSIdentityProviderHelper {  
    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 `AWSIdentityProvider` 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 =  
    AWS Cognito Credentials Provider(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 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 must extend `CognitoAWSCredentials` and override the `RefreshIdentity` method to retrieve the user identity id and token from your backend and return them. Following 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 must extend `CognitoAWSCredentials` and override the `RefreshIdentity` method to retrieve the user identity id and token from your backend and return them. Following is a basic 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. This 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 (for example, 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 (for example, 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 that 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 preceding steps, 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_ACESSTOKEN");
```

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 checks the `logins` map. If the map is not empty and has a key with developer provider name, then call your backend. Otherwise, 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 [AWS Cognito Credentials Provider Helper](#) `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 (for example, you are not providing tokens for multiple providers in your logins dictionary), 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 [AWSIdentityProviderHelper](#) `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. 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 `AmazonCognitoEnhancedIdentityProvider` 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. 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

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(identityDidChange:)  
                                         name:AWSCognitoIdentityIdChangedNotification  
                                         object:nil];  
  
-(void)identityDidChange:(NSNotification*)notification {
```

```
NSDictionary *userInfo = notification.userInfo;
 NSLog(@"identity changed from %@ to %@",
 [userInfo objectForKey:AWSIdentityNotificationPreviousId],
 [userInfo objectForKey:AWSIdentityNotificationNewId]);
}
```

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:AWSIdentityIdChangedNotification
    object:nil)

func identityDidChange(notification: NSNotification!) {
    if let userInfo = notification.userInfo as? [String: AnyObject] {
        print("identity changed from: \(userInfo[AWSIdentityNotificationPreviousId])"
            to: \(userInfo[AWSIdentityNotificationNewId]))"
    }
}
```

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.IdentityChangedEventArgs 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.IdentityChangedEventArgs 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 makes it possible to sync application-related user data across devices. Amazon Cognito Sync can synchronize user profile data across mobile devices and the web without using your own backend. The client libraries cache data locally so that your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data. If you set up push sync, you can 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](#)
- [Synchronizing data across clients](#)
- [Handling event callbacks](#)
- [Implementing push synchronization](#)
- [Implementing Amazon Cognito Sync streams](#)
- [Customizing workflows with Amazon Cognito Events](#)

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.

Set up an identity pool in Amazon Cognito

Amazon Cognito Sync requires an Amazon Cognito identity pool to provide user identities. Before you use Amazon Cognito Sync you must first set up an identity pool. To create an identity pool and install the SDK, see [Getting started with Amazon Cognito identity pools](#).

Store and sync data

After you have set up your identity pool and installed the SDK, you can start storing and syncing data between devices. For more information, see [Synchronizing data across clients](#).

Synchronizing data across clients

 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.

With Amazon Cognito, you can save user data in datasets that contain key-value pairs. Amazon Cognito associates this data with an identity in your identity pool so that your app can access it 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. When your app reads and writes keys, it communicates with this local cache . This communication guarantees that all changes you make 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 must first create a credentials provider. The credentials provider acquires temporary AWS credentials to make it possible for your app to access your AWS resources. You also must import the necessary 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](#).
2. Import the Amazon Cognito package as follows: `import com.amazonaws.mobileconnectors.cognito.*;`
3. Initialize Amazon Cognito Sync. Pass in the Android app context, the identity pool ID, an AWS Region, and an initialized Amazon Cognito credentials provider as follows:

```
CognitoSyncManager client = new CognitoSyncManager(  
    getApplicationContext(),  
    Regions.YOUR_REGION,  
    credentialsProvider);
```

iOS - Objective-C

1. Create a credentials provider, following the instructions in [Getting credentials](#).
2. Import AWSCore and Cognito, and initialize AWSCognito as follows:

```
#import <AWSSiOSSDKv2/AWSCore.h>  
#import <AWSCognitoSync/Cognito.h>  
  
AWSCognito *syncClient = [AWSCognito defaultCognito];
```

3. If you're using CocoaPods, replace <AWSiOSSDKv2/AWSCore.h> with AWSCore.h. Follow the same syntax for the Amazon Cognito import.

iOS - Swift

1. Create a credentials provider, following the instructions in [Getting credentials](#).
2. Import and initialize AWSCognito as follows:

```
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](#).
4. Initialize the Sync Manager as follows:

```
var syncManager = new AWS.CognitoSyncManager();
```

Unity

1. Create an instance of CognitoAWS Credentials, following the instructions in [Getting credentials](#).
2. Create an instance of CognitoSyncManager. Pass the CognitoAwsCredentials object and a AmazonCognitoSyncConfig, and include at least the Region set, as follows:

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =
REGION };
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Xamarin

1. Create an instance of `CognitoAWSCredentials`, following the instructions in [Getting credentials](#).
2. Create an instance of `CognitoSyncManager`. Pass the `CognitoAwsCredentials` object and a `AmazonCognitoSyncConfig`, and include at least the Region set, as follows:

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint = REGION };
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Understanding datasets

Amazon Cognito organizes user profile data 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 that you can synchronize. Read and write operations performed on a dataset only affect the local store until the synchronize method is invoked. Amazon Cognito identifies a dataset by a unique string. You can create a new dataset or open an existing one as follows.

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 as follows:

```
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 as follows:

```
[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 as follows: 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");
```

To delete a key from a dataset, use `Remove` as follows:

```
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 as follows:

```
dataset.Delete();
```

```
dataset.SynchronizeAsync();
```

Reading and writing data in datasets

Amazon Cognito datasets function as dictionaries, with values accessible by key. You can read, add, or modify keys and values of a dataset just as if the dataset were a dictionary, as shown in the following examples.

Note that values you write 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.string(forKey:"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

To remove keys from a dataset, use the `remove` method as follows:

```
dataset.remove("myKey");
```

iOS - Objective-C

To delete a key from a dataset, use `removeObjectForKey` as follows:

```
[dataset removeObjectForKey:@"myKey"];
```

iOS - Swift

To delete a key from a dataset, use `removeObjectForKey` as follows:

```
dataset removeObjectForKey("myKey")
```

Unity

To delete a key from a dataset, use `Remove` as follows:

```
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 event callbacks](#).

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 event callbacks](#).

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 event callbacks](#).

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 event callbacks](#).

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 event callbacks](#).

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 event callbacks](#).

Handling event 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.

As an Amazon Cognito Sync developer, you can implement various callbacks to handle different synchronization events and scenarios. The `SyncCallback` interface in the Android SDK configures notifications about dataset synchronization, including `onSuccess()` when a dataset is successfully downloaded, `onFailure()` when an exception occurs, and `onConflict()` to resolve conflicts between local and remote data.

In the iOS SDK, you can register for similar notifications like `AWSCognitoDidStartSynchronizeNotification` and set handlers like the `AWSCognitoRecordConflictHandler` for conflict resolution. The JavaScript, Unity, and Xamarin platforms have analogous callback mechanisms. When you implement these callbacks, your application can gracefully handle the various synchronization events and scenarios that can occur when using Amazon Cognito Sync.

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) {
        AWSognitoDataset *merged = [[AWSognito 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) {
        AWSognitoDataset *merged = [[AWSognito 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.

AWSognitoDidStartSynchronizeNotification

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: AWSConflictingConflict?) ->  
    AWSConflictingResolvedConflict? in  
    return conflict.resolveWithLocalRecord()  
}
```

Or at the dataset level:

```
dataset.conflictHandler = {  
    (datasetName: String?, conflict: AWSConflictingConflict?) ->  
    AWSConflictingResolvedConflict? in  
    return conflict.resolveWithLocalRecord()  
}
```

Dataset Deleted Handler

When a dataset is deleted, the Amazon Cognito client uses the `AWSConflictingDatasetDeletedHandler` to confirm whether the local cached copy of the dataset should be deleted too. If no `AWSConflictingDatasetDeletedHandler` is implemented, the local data will be purged automatically. Implement an `AWSConflictingDatasetDeletedHandler` 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 = AWSIdentityProvider.defaultIdentityProvider().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 = AWSIdentityProvider.defaultIdentityProvider().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(); //Asynchronously 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(); //Asynchronously fetch the dataset
    }
}
```

```
}

// returning true allows the Synchronize to continue and false stops it
return false;
}
```

Implementing push synchronization

⚠ 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 AWS Identity and Access Management console, configure your IAM roles to have full Amazon SNS access, or create a new role that has full Amazon SNS access. The following example role trust policy grants Amazon Cognito Sync a limited ability to assume an IAM role. Amazon Cognito Sync can only assume the role when it does so on behalf of both the identity pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "cognito-sync.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole",  
            "Condition": {  
                "StringEquals": {  
                    "AWS:SourceAccount": "123456789012"  
                },  
                "ArnLike": {  
                    "AWS:SourceArn": "arn:aws:cognito-identity:us-  
east-1:123456789012:identitypool/us-east-1:177a950c-2c08-43f0-9983-28727EXAMPLE"  
                }  
            }  
        }  
    ]  
}
```

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 = AWSIdentityProvider.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 AWS Cognito 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.

Implementing Amazon Cognito Sync streams

A 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 1 MB, Amazon Cognito includes a presigned Amazon S3 URL 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 Amazon Cognito Sync can no longer assume the role, you turn off the Amazon Cognito streams. You must either recreate the Kinesis stream or fix the role, and then you must turn on the stream again.

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.

Customizing workflows with Amazon Cognito Events

A 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.

i 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.

Note

When configuring a Amazon Cognito Sync trigger outside of the console, you must add Lambda resource-based permissions to allow Amazon Cognito to invoke the function.

You can add this permission from the Lambda console (see [Using resource-based policies for AWS Lambda](#)) or by using the Lambda [AddPermission](#) operation.

Example Lambda Resource-Based Policy

The following AWS Lambda resource-based policy grants Amazon Cognito a limited ability to invoke a Lambda function. Amazon Cognito can only invoke the function on behalf of the identity pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

```
{  
    "Version": "2012-10-17",  
    "Id": "default",  
    "Statement": [  
        {  
            "Sid": "lambda-allow-cognito-my-function",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "cognito-sync.amazonaws.com"  
            },  
            "Action": "lambda:InvokeFunction",  
            "Resource": "<your Lambda function ARN>",  
            "Condition": {  
                "StringEquals": {  
                    "AWS:SourceAccount": "<your account number>"  
                },  
                "ArnLike": {  
                    "AWS:SourceArn": "<your identity pool ARN>"  
                }  
            }  
        }  
    ]
```

```
    }  
}  
}
```

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 programming pattern that service provider interfaces use. Amazon Cognito provides input to your Lambda function in the following JSON format.

```
{
```

```
"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 to have the same format as the input.

When you write functions for the Sync Trigger event, observe the following:

- When Amazon Cognito calls your Lambda function during `UpdateRecords`, the function must respond within 5 seconds. If it doesn't, the Amazon Cognito Sync service generates a `LambdaSocketTimeoutException` exception. You can't increase this timeout value.
- If you get a `LambdaThrottledException` exception, try the sync operation again to update the records.
- Amazon Cognito provides all the records present in the dataset as input to the function.
- Records that the app user updates have the `op` field set as `replace`. The deleted records have `op` field set as `remove`.
- You can modify any record, even if the app user doesn't update the record.
- All the fields except the `datasetRecords` are read-only. Do not change them. If you change these fields, you can't update the records.
 - To modify the value of a record, 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, add a new record to the `datasetRecords` array.

- Amazon Cognito ignores any omitted record in the response when Amazon Cognito updates the record.

Sample Lambda function

The following sample Lambda function shows 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](#)
- [Identity and access management for Amazon Cognito](#)
- [Logging and monitoring in Amazon Cognito](#)
- [Compliance validation for Amazon Cognito](#)
- [Resilience in Amazon Cognito](#)
- [Infrastructure security in Amazon Cognito](#)
- [Configuration and vulnerability analysis in Amazon Cognito user pools](#)
- [AWS managed policies for Amazon Cognito](#)

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

As a managed service, Amazon Cognito is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection](#) in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Cognito through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by 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.

Amazon Cognito user pools and identity pools have IAM-authenticated, unauthenticated, and token-authorized API operations. Unauthenticated and token-authorized API operations are intended for use by your customers, the end users of your app. Unauthenticated and token-authorized API operations are encrypted at rest and in transit. For more information, see [Amazon Cognito user pools authenticated and unauthenticated API operations](#).

 **Note**

Amazon Cognito encrypts your 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 Amazon Cognito resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How Amazon Cognito works with IAM](#)
- [Identity-based policy examples for Amazon Cognito](#)
- [Troubleshooting Amazon Cognito identity and access](#)
- [Using service-linked roles for Amazon Cognito](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Cognito.

Service user – If you use the Amazon Cognito service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon 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 Amazon Cognito, see [Troubleshooting Amazon Cognito identity and access](#).

Service administrator – If you're in charge of Amazon Cognito resources at your company, you probably have full access to Amazon Cognito. It's your job to determine which Amazon Cognito features and resources your service users 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 Amazon Cognito, see [How Amazon Cognito works with IAM](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Cognito. To view example Amazon Cognito identity-based policies that you can use in IAM, see [Identity-based policy examples for Amazon Cognito](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might 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 [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [AWS Multi-factor authentication in IAM](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using

credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

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 [Use cases for IAM users](#) 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. To temporarily assume an IAM role in the AWS Management Console, you can [switch from a user to an IAM role \(console\)](#). 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 [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity

is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **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 [Cross account resource access in IAM](#) 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.
 - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
 - **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create 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 AWS 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 [Use an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. 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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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 [Define custom IAM permissions with customer managed 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 [Choose 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 an 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 [Service control policies](#) in the *AWS Organizations User Guide*.

- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see [Resource control policies \(RCPs\)](#) 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 Amazon Cognito, learn what IAM features are available to use with Amazon Cognito.

IAM features you can use with Amazon Cognito

IAM feature	Amazon Cognito support
Identity-based policies	Yes
Resource-based policies	No

IAM feature	Amazon Cognito support
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	No
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how Amazon 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 Amazon 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 [Define custom IAM permissions with customer managed 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 Amazon Cognito

To view examples of Amazon Cognito identity-based policies, see [Identity-based policy examples for Amazon Cognito](#).

Resource-based policies within Amazon 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 [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for Amazon 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 Amazon Cognito actions, see [Actions defined by Amazon Cognito](#) in the *Service Authorization Reference*.

Policy actions in Amazon 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

When you sign Amazon Cognito API requests with AWS credentials, you can restrict them in an AWS Identity and Access Management (IAM) policy. API requests that you must sign with AWS credentials include server-side sign-in with AdminInitiateAuth, and actions that create, view, or modify your Amazon Cognito resources like UpdateUserPool. For more information about signed API requests, see [Signing AWS API requests](#).

Because Amazon Cognito is a consumer identity product for apps that you want to make available to the public, you have access to the following unsigned APIs. Your app makes these API requests for your users and your prospective users. Some APIs require no prior authorization, like InitiateAuth to start a new authentication session. Some APIs use access tokens or session keys for authorization, like VerifySoftwareToken to complete MFA setup for a user that has an existing authenticated session. An unsigned, authorized Amazon Cognito user pools API supports a Session or AccessToken parameter in the request syntax as displayed in the [Amazon Cognito API Reference](#). An unsigned Amazon Cognito Identity API supports an IdentityId parameter as displayed in the [Amazon Cognito Federated Identities API Reference](#).

For more information about the authorization models and roles of Amazon Cognito user pools API operations, see [Amazon Cognito user pools authenticated and unauthenticated API operations](#).

Amazon Cognito identity pools API operations

- GetId
- GetOpenIdToken
- GetCredentialsForIdentity

- `UnlinkIdentity`

Amazon Cognito user pools API operations

- `AssociateSoftwareToken`
- `ChangePassword`
- `ConfirmDevice`
- `ConfirmForgotPassword`
- `ConfirmSignUp`
- `DeleteUser`
- `DeleteUserAttributes`
- `ForgetDevice`
- `ForgotPassword`
- `GetDevice`
- `GetUser`
- `GetUserAttributeVerificationCode`
- `GlobalSignOut`
- `InitiateAuth`
- `ListDevices`
- `ResendConfirmationCode`
- `RespondToAuthChallenge`
- `RevokeToken`
- `SetUserMFAPreference`
- `SetUserSettings`
- `SignUp`
- `UpdateAuthEventFeedback`
- `UpdateDeviceStatus`
- `UpdateUserAttributes`
- `VerifySoftwareToken`
- `VerifyUserAttribute`

To view examples of Amazon Cognito identity-based policies, see [Identity-based policy examples for Amazon Cognito](#).

Policy resources for Amazon 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 user pools

For Amazon Cognito Your User Pools, it is possible to restrict a 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 Amazon 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 Amazon Cognito identity-based policies, see [Identity-based policy examples for Amazon Cognito](#).

Policy condition keys for Amazon Cognito

Supports service-specific 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 Amazon 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 Amazon Cognito identity-based policies, see [Identity-based policy examples for Amazon Cognito](#).

Access control lists (ACLs) in Amazon 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 Amazon 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.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [Define permissions with ABAC authorization](#) 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 Amazon 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 [Switch from a user to an IAM 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 Amazon Cognito

Supports forward access sessions (FAS): No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for Amazon Cognito

Supports service roles: Yes

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

For details about Amazon Cognito service roles, see [Activate push synchronization](#) and [Implementing push synchronization](#).

 **Warning**

Changing the permissions for a service role might break Amazon Cognito functionality. Edit service roles only when Amazon Cognito provides guidance to do so.

Service-linked roles for Amazon 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 AWS 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 Amazon Cognito service-linked roles, see [Using service-linked roles for Amazon Cognito](#).

Identity-based policy examples for Amazon Cognito

By default, users and roles don't have permission to create or modify Amazon Cognito resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Create IAM policies \(console\)](#) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon Cognito, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon Cognito](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the Amazon Cognito console](#)
- [Allow users to view their own permissions](#)
- [Restricting console access to a specific identity pool](#)
- [Allowing access to specific dataset for all identities in a pool](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies

adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

 **Note**

The original and new versions of the Amazon Cognito console have different underlying behavior when you view and modify your Amazon Cognito resources. If you granted permission to actions under the `cognito-idp` service prefix only when the condition `aws:ViaAWSService` is true, the affected IAM principal could have been effective for Amazon Cognito resources in the original console, but not the new console. To work in the Amazon Cognito console, don't set an `aws:ViaAWSService` condition on Amazon Cognito permissions in your IAM policy.

Using the Amazon 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 Amazon 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 (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 they're trying to perform.

To ensure that users and roles can still use the Amazon Cognito console, also attach the Amazon 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 Amazon Cognito and IAM.

Topics

- [I am not authorized to perform an action in Amazon Cognito](#)
- [I am not authorized to perform iam:PassRole](#)
- [I'm an administrator and want to allow others to access Amazon Cognito](#)
- [I want to allow people outside of my AWS account to access my Amazon Cognito resources](#)

I am not authorized to perform an action in Amazon Cognito

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

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 doesn't 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, the policy for the mateojackson user must be updated to allow access to the *my-example-widget* resource by using the cognito-identity:*GetWidget* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, your policies must be updated to allow you to pass a role to Amazon 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 Amazon Cognito. However, the action requires the service to have permissions that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I'm an administrator and want to allow others to access Amazon Cognito

To allow others to access Amazon Cognito, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in Amazon Cognito. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see [IAM Identities](#) and [Policies and permissions in IAM](#) in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amazon 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 Amazon Cognito supports these features, see [How Amazon Cognito works with IAM](#).
- 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 [Cross account resource access in IAM](#) 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 with a trust policy that permits an AWS service to assume the role. 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:

- **AWSServiceRoleForAmazonCognitoIdpEmailService** – Allows Amazon Cognito user pools service to use your Amazon SES identities for sending email.
- **AWSServiceRoleForAmazonCognitoIdp** – Allows Amazon Cognito user pools to publish events and configure endpoints for your Amazon Pinpoint projects.

AWSServiceRoleForAmazonCognitoIdpEmailService

The AWSServiceRoleForAmazonCognitoIdpEmailService 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 AWSServiceRoleForAmazonCognitoIdpEmailService:

- 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*.

AWSServiceRoleForAmazonCognitoIdp

The `AWSServiceRoleForAmazonCognitoIdp` 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 `AWSServiceRoleForAmazonCognitoIdp`

- 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

You can't edit the `AmazonCognitoIdentity` or `AmazonCognitoIdentityEmailService` service-linked roles in AWS Identity and Access Management. 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. If you delete the role, you only retain entities that Amazon Cognito actively monitors or maintains. Before you can delete `AmazonCognitoIdentity` or `AmazonCognitoIdentityEmailService` service-linked roles, you must do one 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 doesn't use the service-linked role.

Remember to perform the action in each AWS Region with 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 AmazonCognitoldp or AmazonCognitoldpEmailService 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 service-linked roles in all AWS Regions where the service is available. For more information, see [AWS Regions and Endpoints](#).

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 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 Logs** – With CloudWatch Logs, you can send fine-grained logs of user activity to a log group. For example, you can review detailed user activity logs to troubleshoot the delivery of email and SMS messages to your users.
- **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

- [Monitoring and managing costs](#)
- [Exporting logs from Amazon Cognito user pools](#)
- [Tracking quotas and usage in CloudWatch and Service Quotas](#)
- [Amazon Cognito logging in AWS CloudTrail](#)

Monitoring and managing costs

Like with any other AWS service, it's important to understand the effect of your Amazon Cognito configuration and usage on your AWS bill. As part of your preparations for the deployment of user pools to production, set up monitoring and safeguards for activity and resource consumption. When you know where to look and what actions produce additional cost, you can set up precautions against surprises in your bill.

Amazon Cognito charges for the following dimensions of your usage.

- User pool monthly active users (MAUs)—rate varies by [feature plan](#)
- User pool MAUs signed in with OIDC or SAML federation
- Active user pool app clients and request volume for machine to machine (M2M) authorization with client credentials grants
- Purchased usage above default quotas for some categories of user pool APIs

Additionally, features of your user pool like email messages, SMS messages, and Lambda triggers can incur costs in dependent services. For a complete overview, see [Amazon Cognito Pricing](#).

Viewing and anticipating costs

High-volume events like product launches and opening up to new userbases can increase your MAU count and have a cost impact. Estimate the new user count in advance and watch activity as it happens. You might find that you want to accommodate the volume with a purchase of additional quota capacity, or control the volume with additional security measures.

You can view and report on your AWS costs in the [AWS Billing and Cost Management console](#). You can find your most recent charges for Amazon Cognito in the **Billing and payments** section. Under **Bills, Charges by service**, filter on Cognito to view your usage. For more information, see [Viewing your bill](#) in the *AWS Billing User Guide*.

To monitor API request rates, review the **Utilization** metric in the Service Quotas console. For example, client credentials requests display as **Rate of ClientAuthentication requests**. In your bill, these requests are associated with the app client that produced them. With this information, you can equitable allocate costs to the tenants in a [multi-tenant architecture](#).

To get a count of M2M requests for a period of time, you can also send [AWS CloudTrail events to CloudWatch Logs](#) for analysis. Query your CloudTrail events for Token_POST events with a client credentials grant. The following CloudWatch Insights query returns this count.

```
filter eventName = "Token_POST" and @message like '"grant_type": ["client_credentials"]'  
| stats count(*)
```

Managing costs

Amazon Cognito bills based on user count, feature usage, and request volume. The following are some tips to manage cost in Amazon Cognito,

Don't activate inactive users

Typical operations that make a user active are sign-in, sign-up, and password reset. For a more thorough list, see [Monthly active users](#). Amazon Cognito doesn't count inactive users toward your bill. Avoid any operations that set a user active. Instead of the [Admin GetUser](#) API operation, query users with the [ListUsers](#) operation. Don't perform high-volume administrative testing of user pool operations with inactive users.

Link federated users

Users who sign in with a SAML 2.0 or OpenID Connect (OIDC) identity provider have a higher cost than [local users](#). You can [link these users to a local user profile](#). A linked user can sign in as a local

user with the attributes and access that come with their federated user. Users from SAML or OIDC IdPs who, in the course of a month, only sign in with a linked local account are billed as local users.

Manage request rates

If your user pool is approaching the upper limit of your quota, you might consider purchasing additional capacity to handle the volume. You might be able to reduce the volume of requests in your application. For more information, see [Optimize request rates for quota limits](#).

Request a new token only when you need one

Machine to machine (M2M) authorization with client credentials grants can reach a high volume of token requests. Each new token request has an effect on your request-rate quota and the size of your bill. To optimize cost, include token expiration settings and token handling in the design of your applications.

- [Cache access tokens](#) so that when your application requests a new token, it receives a cached version of a previously-issued token. When you implement this method, your caching proxy acts as a guard against applications that request access tokens without awareness of the expiration of previously-acquired tokens. Caching tokens is ideal for short-lived microservices like Lambda functions and Docker containers.
- Implement token-handling mechanisms in your applications that account for token expiration. Don't request a new token until previous tokens are about to expire. As a best practice, refresh tokens at about 75% of the token lifetime. This practice maximizes token duration while ensuring user continuity in your application.

Evaluate the confidentiality and availability needs of each application and configure the user pool app client to issue access tokens with an appropriate validity period. Custom token duration works best with longer-lived APIs and servers that can persistently manage the frequency of requests for credentials.

Delete unused client credentials app clients

M2M authorization bills based on two factors: the rate of token requests and the number of app clients that do client credentials grants. When app clients for M2M authorization aren't in use, delete them or remove their authorization to issue client credentials. For more information about managing app client configuration, see [Application-specific settings with app clients](#).

Manage feature plans

When you choose a [feature plan](#) in a user pool, the billing rate applies to all MAUs in the user pool. If you have users that don't need features that come with a higher-level feature plan, separate them into another user pool.

Exporting logs from Amazon Cognito user pools

You can configure your user pool to send detailed logs of some additional activity to another AWS service, like a CloudWatch log group. These logs are of a finer granularity than those in AWS CloudTrail, and can be useful to troubleshoot your user pool and analyze user sign-in activity with [advanced security features](#). When you want to stream logs of SMS and email notification errors, your user pool sends ERROR-level logs to a CloudWatch log group. When you want to stream logs of user sign-in activity, your user pool sends INFO-level logs to a log group, a Amazon Data Firehose stream, or an Amazon S3 bucket. You can combine both options in a user pool.

Topics

- [Things to know about log export](#)
- [Exporting email and SMS message delivery errors](#)
- [Exporting threat protection user activity logs](#)

Things to know about log export

Cost impact

Amazon Data Firehose, Amazon S3, and CloudWatch Logs incur costs for data ingestion and retrieval. Your logging configuration might affect your AWS bill. For more information, see the following:

- [Vended Logs in Amazon CloudWatch Pricing](#).
- [Amazon Data Firehose pricing](#)
- [Amazon S3 pricing](#)

User-activity log exports contain security assessments and are a function of user pool [advanced security features](#). Amazon Cognito only generates these logs when advanced security features are active. These features increase the cost per monthly active user (MAU) in your user pool. For more information, see [Amazon Cognito Pricing](#).

Best-effort delivery

Delivery of logs from Amazon Cognito is best effort. The volume of logs that your user pool delivers, and your service quotas for CloudWatch Logs, Amazon S3, and Firehose can affect the delivery of logs.

Existing external logs are unaffected

These logging options don't replace or change the following log functions of user pools.

1. CloudTrail logs of routine user activity like sign-up and sign-in.
2. Analysis of user activity at scale with CloudWatch metrics.

Separately, you can also find logs from [Viewing the user pool import results in the CloudWatch console](#) and [Customizing user pool workflows with Lambda triggers](#) in CloudWatch Logs.

Amazon Cognito and Lambda store these logs in different log groups from the ones that you specify for user activity logs.

Applies only to user pools

No log export capabilities exist for identity pools.

Requires user permissions and service-linked role

The AWS principal that sets up log export must have permissions to modify the target resources, as described in the topics that follow. Amazon Cognito creates a [service-linked role](#) on your behalf and assumes the role to deliver logs to the target resource.

For more information about the authorization model for sending logs from Amazon Cognito, see [Enable logging from AWS services](#) in the *Amazon CloudWatch Logs User Guide*.

Log level is exclusive to log type

Message-delivery logs are of the `userNotification` type and of the `ERROR` errorlevel. Advanced security user activity logs are of the `userAuthEvents` type and of the `INFO` errorlevel. You can combine two members of `LogConfigurations`, one for `userNotification` to CloudWatch Logs, and one for `userAuthEvents` to Firehose, Amazon S3, or CloudWatch Logs.

You can't send user-activity logs to multiple destinations. You can't send user-notification logs to any destination other than CloudWatch Logs.

Different configuration options

You can only configure user-notification logs with the Amazon Cognito user pools API or an AWS SDK. You can configure advanced security user-activity logs with the API or in the Amazon Cognito console. To set both, use the API as demonstrated in the example request at [SetLogDeliveryConfiguration](#).

Additional configuration required with large resource-based policies

To send logs to log groups with a resource policy of a size greater than 5120 characters, configure a log group with a path that starts with /aws/vendedlogs. For more information, see [Enabling logging from certain AWS services](#).

Exporting email and SMS message delivery errors

For email and SMS message delivery errors, you can deliver **Error**-level user notification logs from your user pool. When you activate this feature, you can choose the log group where you want Amazon Cognito to send logs. User notification logging is useful when you want to find out the status of email and SMS messages that your user pool delivered with Amazon SNS and Amazon SES. This log export option, unlike [user-activity export](#), doesn't require the Plus feature plan.

You can configure detailed notification logs with the Amazon Cognito user pools API in a [SetLogDeliveryConfiguration](#) API request. You can view the logging configuration of a user pool in a [GetLogDeliveryConfiguration](#) API request. The following is an example request body.

```
{
  "LogConfigurations": [
    {
      "CloudWatchLogsConfiguration": {
        "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:example-user-pool-exported"
      },
      "EventSource": "userNotification",
      "LogLevel": "ERROR"
    }
  ],
  "UserPoolId": "us-west-2_EXAMPLE"
}
```

You must authorize these requests with AWS credentials that have the following permissions.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ManageUserPoolLogs",  
            "Action": [  
                "cognito-idp:SetLogDeliveryConfiguration",  
                "cognito-idp:GetLogDeliveryConfiguration",  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Effect": "Allow"  
        },  
        {  
            "Sid": "CognitoLog",  
            "Action": [  
                "logs>CreateLogDelivery",  
                "logs>GetLogDelivery",  
                "logs>UpdateLogDelivery",  
                "logs>DeleteLogDelivery",  
                "logs>ListLogDeliveries"  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Effect": "Allow"  
        },  
        {  
            "Sid": "CognitoLoggingCWL",  
            "Action": [  
                "logs>PutResourcePolicy",  
                "logs>DescribeResourcePolicies",  
                "logs>DescribeLogGroups"  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Effect": "Allow"  
        }  
    ]  
}
```

The following is an example event from a user pool. This log schema is subject to change. Some fields might be logged with null values.

```
{  
  "eventTimestamp": "1687297330677",  
  "eventSource": "USER_NOTIFICATION",  
  "logLevel": "ERROR",  
  "message": {  
    "details": "String"  
  },  
  "logSourceId": {  
    "userPoolId": "String"  
  }  
}
```

Exporting threat protection user activity logs

User pools with the Plus feature plan and threat protection log user activity events: the details and security assessment of user sign-in, sign-out, and other authentication operations with your user pool. You might want to review user activity logs in your own log-management system, or create an archive. You can export this data to a Amazon CloudWatch Logs log group, an Amazon Data Firehose stream, or an Amazon Simple Storage Service (Amazon S3) bucket. From there, you can ingest this data into other systems that analyze, normalize or otherwise process data in ways that fit it in to your operational processes. To export data of this type, your user pool must be on the Plus feature plan and [advanced security features](#) must be active in your user pool.

With the information in these user activity logs, you can view a profile of user sign-in and account-management activity. By default, Amazon Cognito captures these events to storage that's based in your user pool. The following example is an example event for a user who signed in and was evaluated to have no risk factors. You can retrieve this information with the `AdminListUserAuthEvents` API operation. The following is an example output:

```
{  
  "AuthEvents": [  
    {  
      "EventId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",  
      "EventType": "SignIn",  
      "CreationDate": "2024-06-27T10:49:59.139000-07:00",  
      "EventResponse": "Pass",  
      "EventRisk": {  
        "RiskType": "None",  
        "Score": 0  
      }  
    }  
  ]  
}
```

```
        "RiskDecision": "NoRisk",
        "CompromisedCredentialsDetected": false
    },
    "ChallengeResponses": [
        {
            "ChallengeName": "Password",
            "ChallengeResponse": "Success"
        }
    ],
    "EventContextData": {
        "IpAddress": "192.0.2.1",
        "DeviceName": "Chrome 126, Windows 10",
        "Timezone": "-07:00",
        "City": "null",
        "Country": "United States"
    }
},
"NextToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222#2024-06-27T17:49:59.139Z"
}
```

You can activate log export for user activity in the Amazon Cognito console or with the [SetLogDeliveryConfiguration](#) API operation.

AWS Management Console

1. If you don't already have one that you want to use, create an [S3 bucket](#), [Firehose stream](#), or [CloudWatch log group](#).
2. Sign in to the [Amazon Cognito console](#).
3. Choose **User Pools**.
4. Choose an existing user pool from the list, or [create a user pool](#).
5. Choose the **Advanced security** tab. Locate **Export user activity logs** and choose **Edit**.
6. Under **Logging status**, select the checkbox next to **Activate user activity log export**.
7. Under **Logging destination**, choose the AWS service that you want to handle your logs: **CloudWatch log group**, **Amazon Data Firehose stream**, or **S3 bucket**.
8. Your selection will populate the resource selector with the corresponding resource type. Select a log group, stream, or bucket from the list. You can also select the **Create** button to navigate to the AWS Management Console for the selected service and create a new resource.

9. Select **Save changes**.

API

Choose one type of destination for your user activity logs.

The following is an example SetLogDeliveryConfiguration request body that sets a Firehose stream as the log destination.

```
{  
    "LogConfigurations": [  
        {  
            "EventSource": "userAuthEvents",  
            "FirehoseConfiguration": {  
                "StreamArn": "arn:aws:firehose:us-west-2:123456789012:deliverystream/  
example-user-pool-activity-exported"  
            },  
            "LogLevel": "INFO"  
        }  
    ],  
    "UserPoolId": "us-west-2_EXAMPLE"  
}
```

The following is an example SetLogDeliveryConfiguration request body that sets a Amazon S3 bucket as the log destination.

```
{  
    "LogConfigurations": [  
        {  
            "EventSource": "userAuthEvents",  
            "S3Configuration": {  
                "BucketArn": "arn:aws:s3:::amzn-s3-demo-logging-bucket"  
            },  
            "LogLevel": "INFO"  
        }  
    ],  
    "UserPoolId": "us-west-2_EXAMPLE"  
}
```

The following is an example SetLogDeliveryConfiguration request body that sets a CloudWatch log group as the log destination.

```
{  
    "LogConfigurations": [  
        {  
            "EventSource": "userAuthEvents",  
            "CloudWatchLogsConfiguration": {  
                "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:DOC-  
EXAMPLE-LOG-GROUP"  
            },  
            "LogLevel": "INFO"  
        }  
    ],  
    "UserPoolId": "us-west-2_EXAMPLE"  
}
```

The user that configures log delivery must be a user pool administrator and have the following additional permissions:

Amazon S3

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ManageUserPoolLogs",  
            "Action": [  
                "cognito-idp:SetLogDeliveryConfiguration",  
                "cognito-idp:GetLogDeliveryConfiguration",  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Effect": "Allow"  
        },  
        {  
            "Sid": "ManageLogsS3",  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogDelivery",  
                "s3:PutBucketPolicy",  
                "s3:GetBucketPolicy"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        }
    ]
}
```

CloudWatch Logs

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageUserPoolLogs",
            "Action": [
                "cognito-idp:SetLogDeliveryConfiguration",
                "cognito-idp:GetLogDeliveryConfiguration",
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "ManageLogsCWL",
            "Action": [
                "logs>CreateLogDelivery",
                "logs>GetLogDelivery",
                "logs>UpdateLogDelivery",
                "logs>DeleteLogDelivery",
                "logs>ListLogDeliveries",
                "logs>PutResourcePolicy",
                "logs>DescribeResourcePolicies",
                "logs>DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Amazon Data Firehose

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "ManageUserPoolLogs",
        "Action": [
            "cognito-idp:SetLogDeliveryConfiguration",
            "cognito-idp:GetLogDeliveryConfiguration",
        ],
        "Resource": [
            "*"
        ],
        "Effect": "Allow"
    },
    {
        "Sid": "ManageUserPoolLogsFirehose",
        "Effect": "Allow",
        "Action": [
            "logs>CreateLogDelivery",
            "iam>CreateServiceLinkedRole",
            "firehose:TagDeliveryStream"
        ],
        "Resource": "*"
    }
]
```

The following is an example event from a user pool. This log schema is subject to change. Some fields might be logged with null values.

```
{
    "eventTimestamp": "1687297330677",
    "eventSource": "USER_ACTIVITY",
    "logLevel": "INFO",
    "message": {
        "version": "1",
        "eventId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventType": "SignUp",
        "userSub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "userName": "test-user",
        "userPoolId": "us-west-2_EXAMPLE",
        "clientId": "1example23456789",
        "creationDate": "Wed Jul 17 17:25:55 UTC 2024",
    }
}
```

```
        "eventResponse": "InProgress",
        "riskLevel": "",
        "riskDecision": "PASS",
        "challenges": [],
        "deviceName": "Other, Other",
        "ipAddress": "192.0.2.1",
        "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
        "idpName": "",
        "compromisedCredentialDetected": "false",
        "city": "Seattle",
        "country": "United States",
        "eventFeedbackValue": "",
        "eventFeedbackDate": "",
        "eventFeedbackProvider": ""

    },
    "logSourceId": {
        "userPoolId": "us-west-2_EXAMPLE"
    }
}
```

Tracking quotas and usage in CloudWatch and Service Quotas

You can monitor Amazon Cognito user pools using Amazon CloudWatch or using Service Quotas. You can also monitor identity pools usage in 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 and identity 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 a while. 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](#).

If you are signed in to an AWS account that is set up as a monitoring account in CloudWatch cross-account observability, you can use that monitoring account to visualize service quotas and set alarms for metrics in the source accounts that are linked to that monitoring account. For more information, see [CloudWatch cross-account observability](#).

Topics

- [User pool metrics in CloudWatch](#)
- [Metrics in Service Quotas](#)

User pool metrics in CloudWatch

User pools report user-activity statistics to CloudWatch as metrics. From CloudWatch, you can analyze the volume of authentication activity and quota usage in your user pools. With the information in these metrics, you can set alarms for noteworthy events and adjust your user pool configuration as needed. Where user-activity logging has detailed records of user activity in your user pools, CloudWatch metrics have aggregated statistics and performance indicators.

The following table lists the metrics available for Amazon Cognito user pools. The Amazon CloudWatch metrics namespace for Amazon Cognito is AWS/Cognito. For more information, see [Namespaces in Amazon CloudWatch User Guide](#).

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	<p>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.</p> <p>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.</p> <p>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.</p> <p>Note that this metric is not emitted for User import and User migration cases.</p> <p>Metric dimension: UserPool, UserPoolClient</p> <p>Units: Count</p>

Metric	Description
SignUpThrottles	<p>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.</p> <p>To count the total number of throttled user registration requests, use the Sum statistic for this metric.</p> <p>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.</p> <p>Metric dimension: UserPool, UserPoolClient</p> <p>Units: Count</p>

Metric	Description
SignInSuccesses	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>

Metric	Description
	<p>Metric dimension: UserPool, UserPoolClient</p> <p>Units: Count</p>
SignInThrottles	<p>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.</p> <p>To count the total number of throttled user authentication requests, use the Sum statistic for this metric.</p> <p>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.</p> <p>Requests to refresh Amazon Cognito token is not included in this metric. There is a separate metric for providing Refresh token statistics.</p> <p>Metric dimension: UserPool, UserPoolClient</p> <p>Units: Count</p>

Metric	Description
TokenRefreshSuccesses	<p>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.</p> <p>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.</p> <p>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.</p> <p>Metric dimension: UserPool, UserPoolClient</p> <p>Units: Count</p>

Metric	Description
TokenRefreshThrottles	<p>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.</p> <p>To count the total number of throttled requests to refresh an Amazon Cognito token, use the Sum statistic for this metric.</p> <p>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.</p> <p>Metric dimension: UserPool, UserPoolClient</p> <p>Units: Count</p>

Metric	Description
FederationSuccesses	<p>Provides the total number of successful identity federation requests to the Amazon Cognito user pool. An identity federation is considered successful when Amazon Cognito issues authentication tokens to the user. A successful identity federation request produces a value of 1, whereas an unsuccessful request produces a value of 0. Throttled requests and requests that generate an authorization code but no tokens produce a value of 0.</p> <p>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.</p> <p>Metric dimension: UserPool, UserPoolClient , IdentityProvider</p> <p>Units: Count</p>

Metric	Description
FederationThrottles	<p>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.</p> <p>To count the total number of throttled identity federation requests, use the Sum statistic for this metric.</p> <p>Metric dimension: UserPool, UserPoolClient , IdentityProvider</p> <p>Units: Count</p>
CallCount	<p>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.</p> <p>This metric is available in the Usage nameSpace .</p> <p>The category quota is enforced for each AWS account across all user pools in an account and Region.</p> <p>You can count the total number of calls in a category using the Sum statistic for this metric.</p> <p>Metric dimension: Service, Type, Resource, Class</p> <p>Units: Count</p>

Metric	Description
ThrottleCount	<p>Provides the total number of throttled calls related to a category.</p> <p>This metric is available in the Usage nameSpace .</p> <p>This metric is published at the account level.</p> <p>You can count the total number of calls in a category, using the Sum statistic for this metric.</p> <p>Metric dimension: Service, Type, Resource, Class</p> <p>Units: Count</p>

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.

Dimension	Description
Class	The class of resource being tracked. Amazon Cognito doesn't use the class dimension.

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 set up 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. For more information about tracking usage in CloudWatch, see [Track quota usage](#).

In the Service Quotas console, you can create alarms that alert you when your usage approaches a service quota. To learn how to set up a CloudWatch alarm using the Service Quotas console, see [Service Quotas and CloudWatch alarms](#).

Metrics in Service Quotas

You can view and manage your Amazon Cognito user pools and identity pools quotas from a central location with Service Quotas. You can use the Service Quotas console to view details about a specific quota, monitor quota utilization, and request a quota increase. For some quota types, you can create a CloudWatch alarm to track your quota utilization. To learn more about what Amazon Cognito metrics you can track, see [Track quota usage](#).

To view Amazon Cognito user pools and identity 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, search and choose **Amazon Cognito user pools** or **Amazon Cognito Federated Identities**. The service quota page appears.
4. Select a quota that supports CloudWatch monitoring. For example, choose **Rate of UserAuthentication requests** in Amazon Cognito user pools.
5. Scroll down to **Monitoring**. This section appears only for quotas that support CloudWatch monitoring.
6. In **Monitoring** you can view current service quota utilization in the graph.
7. In **Monitoring** select either one hour, three hours, twelve hours, one day, three days, or one week.
8. 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.

Amazon Cognito logging in 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 choose to deliver 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 activate 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](#).

Topics

- [Information that Amazon Cognito sends to CloudTrail](#)
- [Analyzing Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights](#)
- [Example Amazon Cognito events](#)

Information that Amazon Cognito sends to CloudTrail

CloudTrail is turned on when you create your AWS 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 CloudTrail trail delivers 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](#)

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](#).

Confidential data in AWS CloudTrail

Because user pools and identity pools process user data, Amazon Cognito obscures some private fields in your CloudTrail events with the value HIDDEN_FOR_SECURITY_REASONS. For examples of fields that Amazon Cognito doesn't populate to events, see [Example Amazon Cognito events](#). Amazon Cognito only obscures some fields that commonly contain user information, like passwords and tokens. Amazon Cognito doesn't perform any automatic detection or masking of personally-identifying information that you populate to non-private fields in your API requests.

User pool events

Amazon Cognito supports logging for all of the actions listed on the [User pool actions](#) page as events in CloudTrail log files. Amazon Cognito logs user pool events to CloudTrail as *management events*.

The eventType field in a Amazon Cognito user pools CloudTrail entry tells you whether your app made the request to the [Amazon Cognito user pools API](#) or to an [endpoint that serves resources for OpenID Connect, SAML 2.0, or managed login pages](#). API requests have an eventType of AwsApiCall and endpoint requests have an eventType of AwsServiceEvent.

Amazon Cognito logs the following requests to your managed login services as events in CloudTrail.

Managed login events in CloudTrail

Operation	Description
Login_GET , CognitoAuthentication	A user views or submits credentials to your Login endpoint .
OAuth2_Authorize_GET , Beta_Auth orize_GET	A user views your Authorize endpoint .

Operation	Description
OAuth2Response_GET , OAuth2Response_POST	A user submits an IdP token to your /oauth2/idpresponse endpoint.
SAML2Response_POST , Beta_SAML2Response_POST	A user submits an IdP SAML assertion to your /saml2/idpresponse endpoint.
Login_OIDC_SAML_POST	A user enters a username at your Login endpoint and matches with an IdP identifier .
Token_POST , Beta_Token_POST	A user submits an authorization code to your Token endpoint .
Signup_GET , Signup_POST	A user submits sign-up information to your /signup endpoint.
Confirm_GET , Confirm_POST	A user submits a confirmation code in the hosted UI.
Resendcode_POST	A user submits a request to resend a confirmation code in the hosted UI.
ForgotPassword_GET , ForgotPassword_POST	A user submits a request to reset their password to your /forgotPassword endpoint.
ConfirmForgotPassword_GET , ConfirmForgotPassword_POST	A user submits a code to your /confirmForgotPassword endpoint that confirms their ForgotPassword request.
ResetPassword_GET , ResetPassword_POST	A user submits a new password in the hosted UI.
Mfa_GET, Mfa_POST	A user submits a multi-factor authentication (MFA) code in the hosted UI.
MfaOption_GET , MfaOption_POST	A user chooses their preferred method for MFA in the hosted UI.

Operation	Description
MfaRegister_GET , MfaRegister_POST	A user submits a multi-factor authentication (MFA) code in the hosted UI when registering the MFA.
Logout	A user signs out at your /logout endpoint.
SAML2Logout_POST	A user signs out at your /saml2/logout endpoint.
Error_GET	A user views an error page in the hosted UI.
UserInfo_GET , UserInfo_POST	A user or IdP exchanges information with your userInfo endpoint .
Confirm_With_Link_GET	A user submits a confirmation based on a link that Amazon Cognito sent in an email message.
Event_Feedback_GET	A user submits feedback to Amazon Cognito about an advanced security features event.
login_POST	A user submits credentials to your Login endpoint .
login_continue_POST	A user who has already signed in one time chooses to sign in again.
selectChallenge_POST	A user responds to an authentication challenge after they submit their username or credentials.
confirmUser_GET	A user opens the link in a confirmation or verification email message .
mfa_back_POST	A user chooses the Back button after an MFA prompt.
mfa_options_POST	A user selects an MFA option.

Operation	Description
mfa_phone_register_POST	A user submits a phone number to register as a MFA factor. This operation causes Amazon Cognito to send an MFA code to their phone number.
mfa_phone_verify_POST	A user submits an MFA code sent to their phone number.
mfa_phone_resendcode_POST	A user submits a request to resend a MFA code to their phone number.
mfa_totp_POST	A user submits a TOTP MFA code.
signup_POST	A user submits information to your /signup managed login page.
signup_confirm_POST	A user submits a confirmation code from an email or SMS message.
verifyCode_POST	A user submits a one-time password (OTP) for passwordless authentication.
passkeys_add_POST	A user submits a request to register a new passkey credential.
passkeys_add_GET	A user navigates to the page where they can register a passkey.
login_passkey_POST	A user signs in with a passkey.

 **Note**

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.

Identity pools events

Data events

Amazon Cognito logs the following Amazon Cognito Identity events to CloudTrail as *data events*.

[Data events](#) are high-volume data-plane API operations that CloudTrail doesn't log by default.

Additional charges apply for data events.

- [GetCredentialsForIdentity](#)
- [GetId](#)
- [GetOpenIdToken](#)
- [GetOpenIdTokenForDeveloperIdentity](#)
- [UnlinkIdentity](#)

To generate CloudTrail logs for these API operations, you must activate data events in your trail and choose event selectors for **Cognito identity pools**. For more information, see [Logging data events for trails](#) in the *AWS CloudTrail User Guide*.

You can also add identity pools event selectors to your trail with the following CLI command.

```
aws cloudtrail put-event-selectors --trail-name <trail name> --advanced-event-selectors
 \
"{
  \"Name\": \"Cognito Selector\",
  \"FieldSelectors\": [
    {
      \"Field\": \"eventCategory\",
      \"Equals\": [
        \"Data\"
      ],
      \"Field\": \"resources.type\",
      \"Equals\": [
        \"AWS::Cognito::IdentityPool\"
      ]
    }
  ]
}"
```

Management events

Amazon Cognito logs the remainder of Amazon Cognito identity pools API operations as *management events*. CloudTrail logs management event API operations by default.

For a list of the Amazon Cognito identity pools API operations that Amazon Cognito logs to CloudTrail, see the [Amazon Cognito identity pools API Reference](#).

Amazon Cognito Sync

Amazon Cognito logs all Amazon Cognito Sync API operations as management events. For a list of the Amazon Cognito Sync API operations that Amazon Cognito logs to CloudTrail, see the [Amazon Cognito Sync API Reference](#).

Analyzing Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights

You can search and analyze your Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights. 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
```

Example Amazon Cognito events

Amazon Cognito logs information to AWS CloudTrail about user authentication activity and administrative management activity. This applies to both user pools and identity pools.

For example, you can see GetId and UpdateIdentityPool events in the same trail, or UpdateAuthEventFeedback and SetRiskConfiguration events. You'll also see user pool logs for hosted UI activity that doesn't correspond to operations in the user pools API. This section has some examples of logs you might see. To understand the CloudTrail event schema for any operation, generate a request for that operation and review the events that it creates in your trail.

A trail can deliver 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.

Topics

- [Example CloudTrail events for a hosted UI sign-up](#)
- [Example CloudTrail event for a SAML request](#)
- [Example CloudTrail events for requests to the token endpoint](#)
- [Example CloudTrail event for CreateIdentityPool](#)
- [Example CloudTrail event for GetCredentialsForIdentity](#)
- [Example CloudTrail event for GetId](#)
- [Example CloudTrail event for GetOpenIdToken](#)
- [Example CloudTrail event for GetOpenIdTokenForDeveloperIdentity](#)
- [Example CloudTrail event for UnlinkIdentity](#)

Example CloudTrail events for a hosted UI sign-up

The following example CloudTrail events demonstrate the information that Amazon Cognito logs when a user signs up through the hosted UI.

Amazon Cognito logs the following event when a new user navigates to the sign-in page for your app.

```
{  
    "eventVersion": "1.08",  
    "userIdentity":  
    {  
        "accountId": "123456789012"  
    },  
    "eventTime": "2022-04-06T05:38:12Z",  
    "eventSource": "cognito-idp.amazonaws.com",  
    "eventName": "Login_GET",  
    "awsRegion": "us-west-2",  
    "sourceIPAddress": "192.0.2.1",  
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",  
    "errorCode": ""},
```

```
"errorMessage": "",  
"additionalEventData":  
{  
    "responseParameters":  
    {  
        "status": 200.0  
    },  
    "requestParameters":  
    {  
        "redirect_uri":  
        [  
            "https://www.amazon.com"  
        ],  
        "response_type":  
        [  
            "token"  
        ],  
        "client_id":  
        [  
            "1example23456789"  
        ]  
    }  
},  
"eventID": "382ae09a-151d-4116-8f2b-6ac0a804a38c",  
"readOnly": true,  
"eventType": "AwsServiceEvent",  
"managementEvent": true,  
"recipientAccountId": "123456789012",  
"serviceEventDetails":  
{  
    "serviceAccountId": "111122223333"  
},  
"eventCategory": "Management"  
}
```

Amazon Cognito logs the following event when a new user chooses **Sign up** from the sign-in page for your app.

```
{  
    "eventVersion": "1.08",  
    "userIdentity":  
    {  
        "accountId": "123456789012"
```

```
},
"eventTime": "2022-05-05T23:21:43Z",
"eventSource": "cognito-idp.amazonaws.com",
"eventName": "Signup_GET",
"awsRegion": "us-west-2",
"sourceIPAddress": "192.0.2.1",
"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
"requestParameters": null,
"responseElements": null,
"additionalEventData":
{
    "responseParameters":
    {
        "status": 200
    },
    "requestParameters":
    {
        "response_type":
        [
            "code"
        ],
        "redirect_uri":
        [
            "https://www.amazon.com"
        ],
        "client_id":
        [
            "1example23456789"
        ]
    },
    "userPoolDomain": "mydomain.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_aaaaaaaaaa"
},
"requestID": "7a63e7c2-b057-4f3d-a171-9d9113264fff",
"eventID": "5e7b27a0-6870-4226-adb4-f86cd51ac5d8",
"readOnly": true,
"eventType": "AwsServiceEvent",
"managementEvent": true,
"recipientAccountId": "123456789012",
"serviceEventDetails":
{
    "serviceAccountId": "111122223333"
},
"eventCategory": "Management"
```

}

Amazon Cognito logs the following event when a new user chooses a username, enters an email address, and chooses a password from the sign-in page for your app. Amazon Cognito doesn't log identifying information about the user's identity to CloudTrail.

```
{  
    "eventVersion": "1.08",  
    "userIdentity":  
    {  
        "accountId": "123456789012"  
    },  
    "eventTime": "2022-05-05T23:22:05Z",  
    "eventSource": "cognito-idp.amazonaws.com",  
    "eventName": "Signup_POST",  
    "awsRegion": "us-west-2",  
    "sourceIPAddress": "192.0.2.1",  
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",  
    "requestParameters": null,  
    "responseElements": null,  
    "additionalEventData":  
    {  
        "responseParameters":  
        {  
            "status": 302  
        },  
        "requestParameters":  
        {  
            "password":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ],  
            "requiredAttributes[email)":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ],  
            "response_type":  
            [  
                "code"  
            ],  
            "_csrf":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ]  
        }  
    }  
}
```

```
[  
    "redirect_uri":  
    [  
        "https://www.amazon.com"  
    ],  
    "client_id":  
    [  
        "1example23456789"  
    ],  
    "username":  
    [  
        "HIDDEN_DUE_TO_SECURITY_REASON"  
    ]  
},  
    "userPoolDomain": "mydomain.us-west-2.amazonaws.com",  
    "userPoolId": "us-west-2_aaaaaaaaa"  
,  
    "requestID": "9ad58dd8-3517-4aa8-96a5-d17a01df9eb4",  
    "eventID": "c75eb7a5-eb8c-43d1-8331-f4412e756e69",  
    "readOnly": false,  
    "eventType": "AwsServiceEvent",  
    "managementEvent": true,  
    "recipientAccountId": "123456789012",  
    "serviceEventDetails":  
    {  
        "serviceAccountId": "111122223333"  
    },  
    "eventCategory": "Management"  
}]
```

Amazon Cognito logs the following event when a new user accesses the user confirmation page in the hosted UI after they sign up.

```
{  
    "eventVersion": "1.08",  
    "userIdentity":  
    {  
        "accountId": "123456789012"  
    },  
    "eventTime": "2022-05-05T23:22:06Z",  
    "eventSource": "cognito-idp.amazonaws.com",  
    "eventName": "Confirm_GET",  
    "awsRegion": "us-west-2",  
}
```

```
"sourceIPAddress": "192.0.2.1",
"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
"requestParameters": null,
"responseElements": null,
"additionalEventData":
{
    "responseParameters":
    {
        "status": 200
    },
    "requestParameters":
    {
        "response_type":
        [
            "code"
        ],
        "redirect_uri":
        [
            "https://www.amazon.com"
        ],
        "client_id":
        [
            "1example23456789"
        ]
    },
    "userPoolDomain": "mydomain.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_aaaaaaaaa"
},
"requestID": "58a5b170-3127-45bb-88cc-3e652d779e0b",
"eventID": "7f87291a-6d50-409a-822f-e3a5ec7e60da",
"readOnly": false,
"eventType": "AwsServiceEvent",
"managementEvent": true,
"recipientAccountId": "123456789012",
"serviceEventDetails":
{
    "serviceAccountId": "111122223333"
},
"eventCategory": "Management"
}
```

Amazon Cognito logs the following event when, in the user confirmation page in the hosted UI, a user enters a code that Amazon Cognito sent them in an email message.

```
{  
    "eventVersion": "1.08",  
    "userIdentity":  
    {  
        "accountId": "123456789012"  
    },  
    "eventTime": "2022-05-05T23:23:32Z",  
    "eventSource": "cognito-idp.amazonaws.com",  
    "eventName": "Confirm_POST",  
    "awsRegion": "us-west-2",  
    "sourceIPAddress": "192.0.2.1",  
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",  
    "requestParameters": null,  
    "responseElements": null,  
    "additionalEventData":  
    {  
        "responseParameters":  
        {  
            "status": 302  
        },  
        "requestParameters":  
        {  
            "confirm":  
            [  
                ""  
            ],  
            "deliveryMedium":  
            [  
                "EMAIL"  
            ],  
            "sub":  
            [  
                "704b1e47-34fe-40e9-8c41-504997494531"  
            ],  
            "code":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ],  
            "destination":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ],  
            "response_type":  
        }  
    }  
}
```

```
[  
    "code"  
,  
    "_csrf":  
    [  
        "HIDDEN_DUE_TO_SECURITY_REASONS"  
,  
        "cognitoAsfData":  
        [  
            "HIDDEN_DUE_TO_SECURITY_REASONS"  
,  
            "redirect_uri":  
            [  
                "https://www.amazon.com"  
,  
                "client_id":  
                [  
                    "1example23456789"  
,  
                    "username":  
                    [  
                        "HIDDEN_DUE_TO_SECURITY_REASONS"  
                    ]  
                ],  
                "userPoolDomain": "mydomain.us-west-2.amazonaws.com",  
                "userPoolId": "us-west-2_aaaaaaaaaa"  
,  
                "requestID": "9764300a-ed35-4f87-8a0f-b18b3fe2b11e",  
                "eventID": "e24ac6e5-2f70-4c6e-ad4e-2f08a547bb36",  
                "readOnly": false,  
                "eventType": "AwsServiceEvent",  
                "managementEvent": true,  
                "recipientAccountId": "123456789012",  
                "serviceEventDetails":  
                {  
                    "serviceAccountId": "111122223333"  
                },  
                "eventCategory": "Management"  
            ]  
}
```

Example CloudTrail event for a SAML request

Amazon Cognito logs the following event when a user who has authenticated with your SAML IdP submits the SAML assertion to your /saml2/idpresponse endpoint.

```
{  
    "eventVersion": "1.08",  
    "userIdentity":  
    {  
        "accountId": "123456789012"  
    },  
    "eventTime": "2022-05-06T00:50:57Z",  
    "eventSource": "cognito-idp.amazonaws.com",  
    "eventName": "SAML2Response_POST",  
    "awsRegion": "us-west-2",  
    "sourceIPAddress": "192.0.2.1",  
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",  
    "requestParameters": null,  
    "responseElements": null,  
    "additionalEventData":  
    {  
        "responseParameters":  
        {  
            "status": 302  
        },  
        "requestParameters":  
        {  
            "RelayState":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ],  
            "SAMLResponse":  
            [  
                "HIDDEN_DUE_TO_SECURITY_REASON"  
            ]  
        },  
        "userPoolDomain": "mydomain.us-west-2.amazoncognito.com",  
        "userPoolId": "us-west-2_aaaaaaaaa"  
    },  
    "requestID": "4f6f15d1-c370-4a57-87f0-aac4817803f7",  
    "eventID": "9824b50f-d9d1-4fb8-a2c1-6aa78ca5902a",  
    "readOnly": false,  
    "eventType": "AwsServiceEvent",  
    "managementEvent": true,  
}
```

```
"recipientAccountId": "625647942648",
"serviceEventDetails":
{
    "serviceAccountId": "111122223333"
},
"eventCategory": "Management"
}
```

Example CloudTrail events for requests to the token endpoint

The following are example events from requests to the [Token endpoint](#).

Amazon Cognito logs the following event when a user who has authenticated and received an authorization code submits the code to your /oauth2/token endpoint.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-12T22:12:30Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Token_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
        "responseParameters": {
            "status": 200
        },
        "requestParameters": {
            "code": [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "grant_type": [

```

```
        "authorization_code"
    ],
    "redirect_uri":
    [
        "https://www.amazon.com"
    ],
    "client_id":
    [
        "1example23456789"
    ]
},
"userPoolDomain": "mydomain.us-west-2.amazonawscognito.com",
"userPoolId": "us-west-2_aaaaaaaaaa"
},
"requestID": "f257f752-cc14-4c52-ad5b-152a46915238",
"eventID": "0bd1586d-cd3e-4d7a-abaf-fd8bfc3912fd",
"readOnly": false,
"eventType": "AwsServiceEvent",
"managementEvent": true,
"recipientAccountId": "123456789012",
"serviceEventDetails":
{
    "serviceAccountId": "111122223333"
},
"eventCategory": "Management"
}
```

Amazon Cognito logs the following event when your backend system submits a `client_credentials` request for an access token to your `/oauth2/token` endpoint.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-12T21:07:05Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Token_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
```

```
"responseElements": null,
"additionalEventData":
{
    "responseParameters":
    {
        "status": 200
    },
    "requestParameters":
    {
        "grant_type":
        [
            "client_credentials"
        ],
        "client_id":
        [
            "1example23456789"
        ]
    },
    "userPoolDomain": "mydomain.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_aaaaaaaaaa"
},
"requestID": "4f871256-6825-488a-871b-c2d9f55caff2",
"eventID": "473e5cbc-a5b3-4578-9ad6-3dfdc8a6d34",
"readOnly": false,
"eventType": "AwsServiceEvent",
"managementEvent": true,
"recipientAccountId": "123456789012",
"serviceEventDetails":
{
    "serviceAccountId": "111122223333"
},
"eventCategory": "Management"
}
```

Amazon Cognito logs the following event when your app exchanges a refresh token for a new ID and access token with your /oauth2/token endpoint.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
}
```

```
"eventTime": "2022-05-12T22:16:40Z",
"eventSource": "cognito-idp.amazonaws.com",
"eventName": "Token_POST",
"awsRegion": "us-west-2",
"sourceIPAddress": "192.0.2.1",
"userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
"requestParameters": null,
"responseElements": null,
"additionalEventData":
{
    "responseParameters":
    {
        "status": 200
    },
    "requestParameters":
    {
        "refresh_token":
        [
            "HIDDEN_DUE_TO_SECURITY_REASONS"
        ],
        "grant_type":
        [
            "refresh_token"
        ],
        "client_id":
        [
            "1example23456789"
        ]
    },
    "userPoolDomain": "mydomain.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_aaaaaaaaaa"
},
"requestID": "2829f0c6-a3a9-4584-b046-11756dfe8a81",
"eventID": "12bd3464-59c7-44fa-b8ff-67e1cf092018",
"readOnly": false,
"eventType": "AwsServiceEvent",
"managementEvent": true,
"recipientAccountId": "123456789012",
"serviceEventDetails":
{
    "serviceAccountId": "111122223333"
},
"eventCategory": "Management"
```

}

Example CloudTrail event for CreateIdentityPool

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": "0000000000000000"  
        }  
    },  
    "responseElements": {  
        "identityPoolName": "TestPool",  
        "identityPoolId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",  
        "allowUnauthenticatedIdentities": true,  
        "supportedLoginProviders": {  
            "graph.facebook.com": "0000000000000000"  
        }  
    },  
    "requestID": "15cc73a1-0780-460c-91e8-e12ef034e116",  
    "eventID": "f1d47f93-c708-495b-bff1-cb935a6064b2",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "123456789012"  
}
```

Example CloudTrail event for GetCredentialsForIdentity

The following example is a log entry for a request for the GetCredentialsForIdentity action.

```
{  
    "eventVersion": "1.08",  
    "userIdentity": {  
        "type": "Unknown"  
    },  
    "eventTime": "2023-01-19T16:55:08Z",  
    "eventSource": "cognito-identity.amazonaws.com",  
    "eventName": "GetCredentialsForIdentity",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "192.0.2.4",  
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off  
command/cognito-identity.get-credentials-for-identity",  
    "requestParameters": {  
        "logins": {  
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaaa":  
"HIDDEN_DUE_TO_SECURITY_REASONS"  
        },  
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"  
    },  
    "responseElements": {  
        "credentials": {  
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",  
            "sessionToken": "aAaAaAaAaAab1111111111EXAMPLE",  
            "expiration": "Jan 19, 2023 5:55:08 PM"  
        },  
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"  
    },  
    "requestID": "659dfc23-7c4e-4e7c-858a-1abce884d645",  
    "eventID": "6ad1c766-5a41-4b28-b5ca-e223ccb00f0d",  
    "readOnly": false,  
    "resources": [{  
        "accountId": "111122223333",  
        "type": "AWS::Cognito::IdentityPool",  
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-  
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"  
    }],  
    "eventType": "AwsApiCall",  
    "managementEvent": false,  
    "recipientAccountId": "111122223333",  
    "eventCategory": "Data"
```

}

Example CloudTrail event for GetId

The following example is a log entry for a request for the `GetId` action.

```
{  
    "eventVersion": "1.08",  
    "userIdentity": {  
        "type": "Unknown"  
    },  
    "eventTime": "2023-01-19T16:55:05Z",  
    "eventSource": "cognito-identity.amazonaws.com",  
    "eventName": "GetId",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "192.0.2.4",  
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off  
command/cognito-identity.get-id",  
    "requestParameters": {  
        "identityPoolId": "us-east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE",  
        "logins": {  
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaaa":  
"HIDDEN_DUE_TO_SECURITY_REASONS"  
        }  
    },  
    "responseElements": {  
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"  
    },  
    "requestID": "dc28def9-07c8-460a-a8f3-3816229e6664",  
    "eventID": "c5c459d9-40ec-41fd-8f6b-57865d5a9975",  
    "readOnly": false,  
    "resources": [{  
        "accountId": "111122223333",  
        "type": "AWS::Cognito::IdentityPool",  
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-  
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"  
    }],  
    "eventType": "AwsApiCall",  
    "managementEvent": false,  
    "recipientAccountId": "111122223333",  
    "eventCategory": "Data"  
}
```

Example CloudTrail event for GetOpenIdToken

The following example is a log entry for a request for the GetOpenIdToken action.

```
{  
    "eventVersion": "1.08",  
    "userIdentity": {  
        "type": "Unknown"  
    },  
    "eventTime": "2023-01-19T16:55:08Z",  
    "eventSource": "cognito-identity.amazonaws.com",  
    "eventName": "GetOpenIdToken",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "192.0.2.4",  
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off  
command/cognito-identity.get-open-id-token",  
    "requestParameters": {  
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",  
        "logins": {  
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaaa":  
"HIDDEN_DUE_TO_SECURITY_REASONS"  
        }  
    },  
    "responseElements": {  
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"  
    },  
    "requestID": "a506ba18-10d7-4fdb-9548-a8187b2e38bb",  
    "eventID": "19ffc1a6-6ed8-4580-a4e1-3062c5ce6457",  
    "readOnly": false,  
    "resources": [{  
        "accountId": "111122223333",  
        "type": "AWS::Cognito::IdentityPool",  
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-  
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"  
    }],  
    "eventType": "AwsApiCall",  
    "managementEvent": false,  
    "recipientAccountId": "111122223333",  
    "eventCategory": "Data"  
}
```

Example CloudTrail event for GetOpenIdTokenForDeveloperIdentity

The following example is a log entry for a request for the GetOpenIdTokenForDeveloperIdentity action.

```
{  
    "eventVersion": "1.08",  
    "userIdentity": {  
        "type": "AssumedRole",  
        "principalId": "AROA1EXAMPLE:johns-AssumedRoleSession",  
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/johns-AssumedRoleSession",  
        "accountId": "111122223333",  
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",  
        "sessionContext": {  
            "sessionIssuer": {  
                "type": "Role",  
                "principalId": "AROA1EXAMPLE",  
                "arn": "arn:aws:iam::111122223333:role/Admin",  
                "accountId": "111122223333",  
                "userName": "Admin"  
            },  
            "attributes": {  
                "creationDate": "2023-01-19T16:53:14Z",  
                "mfaAuthenticated": "false"  
            }  
        }  
    },  
    "eventTime": "2023-01-19T16:55:08Z",  
    "eventSource": "cognito-identity.amazonaws.com",  
    "eventName": "GetOpenIdTokenForDeveloperIdentity",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "27.0.3.154",  
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off  
command/cognito-identity.get-open-id-token-for-developer-identity",  
    "requestParameters": {  
        "tokenDuration": 900,  
        "identityPoolId": "us-east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE",  
        "logins": {  
            "JohnsDeveloperProvider": "HIDDEN_DUE_TO_SECURITY_REASON"  
        }  
    },  
    "responseElements": {  
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"  
    },  
}
```

```
"requestID": "b807df87-57e7-4dd6-b90c-b06f46a61c21",
"eventID": "f26fed91-3340-4d70-91ae-cdf55547b76",
"readOnly": false,
"resources": [
    "accountId": "111122223333",
    "type": "AWS::Cognito::IdentityPool",
    "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "111122223333",
"eventCategory": "Data"
}
```

Example CloudTrail event for UnlinkIdentity

The following example is a log entry for a request for the `UnlinkIdentity` action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "Unknown"
    },
    "eventTime": "2023-01-19T16:55:08Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "UnlinkIdentity",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.4",
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off command/cognito-identity.unlink-identity",
    "requestParameters": {
        "logins": {
            "cognito-idp.us-east-1.amazonaws.com/us-east-1aaaaaaaa": "HIDDEN_DUE_TO_SECURITY_REASONS"
        },
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",
        "loginsToRemove": ["cognito-idp.us-east-1.amazonaws.com/us-east-1aaaaaaaa"]
    },
    "responseElements": null,
    "requestID": "99c2c8e2-9c29-416f-bb17-b650a5cbada9",
    "eventID": "d8e26126-202a-43c2-b458-3f225efaedc7",
    "readOnly": false,
    "resources": [
}
```

```
        "accountId": "111122223333",
        "type": "AWS::Cognito::IdentityPool",
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
    ],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data"
}
```

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](#)

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 same region as the associated user pool.
- 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 AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection](#) in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Cognito through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by 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](#)
- [Shared Responsibility Model](#)

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*.

AWS managed IAM policies that 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 users continue to have the same level of access.

Note

When you create a new identity pool, you can automatically create new roles for authenticated and guest user access. The administrator who creates your identity pool with new IAM roles must also have IAM permissions to create roles.

Identity pools with unauthenticated guest access apply an additional AWS managed policy as a [session policy](#) to unauthenticated users. This AWS managed policy has no intended administrative use. Instead, it limits the scope of permissions that you can apply to guest users in the identity pools [enhanced authentication flow](#). For more information, see [IAM roles](#).

AWS managed IAM policies that Amazon Cognito grants to guest users

- `AmazonCognitoUnAuthedIdentitiesSessionPolicy` - In combination with an inline session policy, limits the permissions that IAM administrators can grant to identity pool guest users. Amazon Cognito automatically applies this policy to guest sessions. For more information, see [The AWS managed session policy for guests](#).

Amazon Cognito updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Cognito since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Cognito [Document history](#) page.

Change	Description	Date
<code>AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy</code> -Change	Amazon Cognito added new actions to permit the use of AWS Key Management Service for unauthenticated (guest) users in identity pools.	October 30, 2024

Change	Description	Date
AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy –Change	Amazon Cognito added new actions to permit the use of Amazon Location Service for unauthenticated (guest) users in identity pools.	August 9, 2024
AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy –New policy	Added an AWS managed policy for privilege scope-down of guest users in identity pools.	July 14, 2023
AmazonCognitoPowerUser and AmazonCognitoReadOnly –Change	Added new permissions to allow power users to view and manage associations of AWS WAF web ACLs to Amazon Cognito user pools. Added new permissions to allow read-only users to view associations of AWS WAF web ACLs to Amazon Cognito user pools.	July 19, 2022

Change	Description	Date
AmazonCognitoPowerUser –Change	<p>Added a new permission to allow Amazon Cognito to call Amazon Simple Email Service PutIdentityPolicy and ListConfigurationSets operations.</p> <p>This change allows Amazon Cognito user pools to update Amazon SES sending authorization policies and to apply Amazon SES configuration sets when you configure email sending in your user pool.</p>	November 17, 2021
AmazonCognitoPowerUser –Change	<p>Added a new permission to allow Amazon Cognito to call Amazon Simple Notification Service's GetSMSandboxAccountStatus operation.</p> <p>This change allows Amazon Cognito 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.</p>	June 1, 2021
Amazon Cognito started tracking changes	Amazon Cognito started tracking changes for its AWS managed policies.	March 1, 2021

Tagging Amazon Cognito resources

A *tag* is a metadata label that you or 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 that you can assign the same tag to resources from different services. This helps you indicate which 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 can activate these tags on the AWS Billing and Cost Management dashboard. AWS uses cost allocation 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 User Guide*.
- Control access to your resources based on the tags that are assigned to them. You can 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 a user to update a user pool 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 Command Line Interface or the Amazon Cognito API to add, edit, or delete tags for both user and identity pools. You can also manage tags for user pools 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 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 keys and values – a-z, A-Z, 0-9, space, and the following characters: _ . : / = + - @
- Keys and values are case sensitive
- Don't use aws : as a prefix for keys; it's reserved for AWS use

Managing tags using the Amazon Cognito console

You can use the Amazon Cognito console to manage the tags that are assigned to your user pools.

To add tags to a user pool

1. Navigate to the [Amazon Cognito console](#). If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or [create a user pool](#).
4. Choose the **Settings** menu and locate the **Tags** tab.
5. Choose **Add tags** to add your first tag. If you have previously assigned tags to this user pool, in **Manage tags**, chose **Add another**.
6. Specify values for **Tag Key** and **Tag Value**.
7. For each additional tag that you want to add, choose **Add another**.
8. When you are finished adding tags, choose **Save changes**.

To tag an identity pool, navigate to the **Identity pools** menu and select or create an identity pool. In the **Identity pool properties** tab, locate **Tags**. Choose **Add tag**.

AWS CLI examples

The AWS CLI provides commands that help you 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, in the format *key=value*.

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, in the format **key=value**.

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. Don't include the tag values. Separate keys with spaces.

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. Don't 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 for up to 30 days after 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
```

Key-value pairs for tags must be in the format `key=value`. If you are adding multiple tags, specify them in a comma-separated list.

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
```

Key-value pairs for tags must be in the format `key=value`. If you are adding multiple tags, specify them in a comma-separated list.

Managing tags 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 has default quotas, formerly referred to as *limits*, for the maximum number of operations that you can perform in your account. Amazon Cognito also has quotas for the maximum number and size of Amazon Cognito resources.

Each Amazon Cognito quota represents a maximum volume of requests in one AWS Region in one AWS account. For example, your apps can make API requests at *up to the Default quota (RPS)* rate for UserAuthentication operations against all of your user pools in US East (N. Virginia). Your apps in Asia Pacific (Tokyo) can produce the same volume of requests against all of your user pools in their own Region. AWS can only grant a quota increase request in one Region at a time. A successful quota increase in US East (N. Virginia) has no effect on your maximum request rate in Asia Pacific (Tokyo).

Topics

- [Understanding API request rate quotas](#)
- [Managing API request rate quotas](#)
- [Amazon Cognito user pools API operation categories and request rate quotas](#)
- [Amazon Cognito identity pools \(federated identities\) API operation request rate quotas](#)
- [Quotas on resource number and size](#)

Understanding API request rate quotas

Quota categorization

Amazon Cognito enforces a maximum request rate for API operations. For more information about the API operations that Amazon Cognito makes available, see the API reference guides for [user pools](#) and [identity pools](#). For user pools, these operations are grouped into categories of common use cases like UserAuthentication or UserCreation. For a list of user pool API operations by category, see [Amazon Cognito user pools API operation categories and request rate quotas](#).

In the [Service Quotas console](#), you can track your quota usage by category user pools and identity pools. If the request rate of your Amazon Cognito user pools or exceeds a quota, you can purchase additional capacity. You can track your user pool quota usage by category and purchase quota increases in the [Service Quotas console](#).

Operation quotas are defined as the maximum number of 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 50 RPS. If multiple operations take place at the same time, each operation within this category can call up to 50 RPS separately or combined.

 **Note**

Category quotas only apply to user pools. Amazon Cognito applies each identity pool quota to a single operation. For both per-category and per-operation request rate quotas, AWS measures the aggregate rate of all requests from all user pools or identity pools in your AWS account in one Region.

Amazon Cognito user pools API operations with special request rate handling

Operation quotas are measured and enforced for the combined total requests at the category level, except for the `AdminRespondToAuthChallenge` and `RespondToAuthChallenge` operations, where special handling rules are applied.

The `UserAuthentication` category includes four operations in the Amazon Cognito user pools API: `AdminInitiateAuth`, `InitiateAuth`, `AdminRespondToAuthChallenge`, and `RespondToAuthChallenge`. Additionally, user authentication in the hosted UI contributes to this quota. The `InitiateAuth` and `AdminInitiateAuth` operations are measured and enforced per category quota. The matching operations `RespondToAuthChallenge` and `AdminRespondToAuthChallenge` are subject to a separate quota that is three times the `UserAuthentication` category limit. This elevated quota accommodates multiple authentication challenges set up in your apps. The quota is sufficient to cover the large majority of use cases. After your app makes up to three responses to authentication challenges, additional requests count toward the `UserAuthentication` category quota. [Multi-factor authentication \(MFA\), device authentication](#), and [custom authentication](#) are all examples of challenge prompts that you might engineer into your user pool.

For example, if your quota for the `UserAuthentication` category is 80 RPS, you can call `RespondToAuthChallenge` or `AdminRespondToAuthChallenge` at a rate up to 240 RPS ($3 * 80$ RPS). If your user pool prompts for four rounds of challenge per authentication and 70 users

sign in per second, then the total RespondToAuthChallenge is 280 RPS (70×4), which is 40 RPS above the quota. The extra 40 RPS is added to 70 InitiateAuth calls, making the total usage of UserAuthentication category 110 RPS ($40 + 70$). Because this value exceeds the category quota set at 80 RPS by 30 RPS, Amazon Cognito throttles requests from your app.

Monthly active users

When Amazon Cognito calculates user pool billing, it charges you a rate for each *monthly active user (MAU)*. Consider your current and projected MAU count in your planning for quota increase requests. A user is counted as a MAU if, within a calendar month, there is an identity operation related to that user. When you [link federated users to local users](#), the MAU count is one plus n , where n is the number of linked identities that have signed in. The activities that make a user active include the following.

- Sign-up or administrative creation of a user
- Sign-in
- Sign-out
- User account confirmation or attribute verification
- Password reset
- Change user attributes, group membership, or MFA preferences
- Query detailed attributes of a user
- User activation, deactivation or deletion

Note

The category *Query detailed attributes of a user* includes the API operation [Admin GetUser](#), but not [ListUsers](#). A detailed user-by-user query in a large user pool can have a significant impact on your AWS bill. To avoid excess charges, collect user data with [ListUsers](#) or store user information in an external database.

You aren't charged for additional sessions by any active user, or for any users that weren't active within a calendar month. In a month where you have changed your user pool feature plan between the available options of *Lite*, *Essentials*, and *Plus*, your bill for that month is computed from the sum of monthly active users (MAUs) in each tier, with each MAU assigned to the highest-priced assigned tier when the user was active. For example:

1. At the beginning of the month, your user pool is on the Plus feature plan.
2. User A signs in on the first day of the month.
3. User B signs in on the first and last days of the month.
4. On the tenth day of the month, you switch your feature plan to Essentials.
5. User C signs in on the last day of the month.

In this scenario, user A and user B are Plus MAUs and user C is an Essentials MAU.

Lite MAU

A user that was active at least once in a month when the user pool was on the Lite feature plan, and was never active when the user pool was on the Essentials or Plus plans.

Essentials MAU

A user that was active at least once in a month when the user pool was on the Essentials feature plan, and was never active when the user pool was on the Plus plan.

Plus MAU

A user that was active at least once in a month when the user pool was on the Plus plan.

For more information, see [User pool feature plans](#).

Managing API request rate quotas

Identify quota requirements

Important

If you increase Amazon Cognito quotas for categories such as `UserAuthentication`, `UserCreation`, or `AccountRecovery`, you may need to increase quotas for other AWS services. For example, messages that Amazon Cognito sends with Amazon Simple Notification Service (Amazon SNS) or Amazon Simple Email Service (Amazon SES) can fail if request rate quotas are insufficient in those services.

To calculate quota requirements, determine how many active users will interact with your application in a specific time period. For example, if you expect your application to sign in

an average of one million active users within an eight-hour period, then you must be able to authenticate an average of 35 users per second.

In addition, if you assume that the average user session is two hours, and you configure tokens to expire after an hour, each user must refresh their tokens once during their session. 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 for 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 must sum up the individual operation call rates at the category level.

Optimize request rates for quota limits

Because increasing API rate limits adds costs to your AWS bill, consider adjustments to your usage model before you request a quota increase. The following are some examples of app architecture that optimizes request rates.

Retry the attempt after a back-off waiting period

You can catch errors with each 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, see [Tools to Build on AWS](#).

Use an 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 JSON web tokens (JWTs) on the client side

Applications must validate JWT tokens before trusting them. You can verify the signature and validity of tokens on the client side without sending API requests 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, see [Verifying a JSON Web Token](#).

Throttle traffic to your web application with a waiting room

If you expect traffic from a large number of users signing in during a time-bound event, such as taking an exam or attending a live event, you can optimize request traffic with self-throttling mechanisms. You can, for example, set up a waiting room where users can stand by until a session is available, allowing you to process requests when you have available capacity. See the [AWS Virtual Waiting Room solution](#) for a reference architecture of a waiting room.

Cache JWTs

Reuse access tokens until they expire. For an example framework with token caching in an API Gateway, see [Managing user pool token expiration and caching](#). Instead of generating API requests to query user information, cache ID tokens until they expire, and read user attributes from the cache.

For more information about working with API request rates in AWS, see [Managing and monitoring API throttling in your workloads](#). For information about optimizing Amazon Cognito operations that add costs to your AWS bill, see [Managing costs](#).

Track quota usage

Amazon Cognito generates `CallCount` and `ThrottleCount` metrics in Amazon CloudWatch for each API operation category 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 `ThrottleCount` metrics with the `Sum` statistic to count the total number of calls in a category. For more information, see [CloudWatch usage metrics](#).

When monitoring service quotas, *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 with CloudWatch. The CloudWatch dashboard displays metrics about every AWS service that you use. With CloudWatch, you can create metric alarms to notify you or change a specific resource that you are monitoring. For more information about CloudWatch metrics, see [Track your CloudWatch usage metrics](#).

Tracking utilization through Service Quotas metrics

Amazon Cognito user pools are integrated with Service Quotas, a console interface to display 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, or set up CloudWatch alarms. After your account has been active for a while, you can view a graph of your resource utilization.

The **Applied account-level quota value** column in the Service Quotas console for [Amazon Cognito user pools](#) and [Amazon Cognito identity pools](#) displays your current quota. The **Utilization** column displays your current rate of quota usage. Adjustable Amazon Cognito user pools requests-per-second (RPS) quotas display their current usage. The Service Quotas console can also navigate you to CloudWatch metrics for a closer look at a selected quota metric. For more information on viewing quotas in the Service Quotas console, see [Viewing Service Quotas](#).

Track monthly active users (MAUs)

The number of monthly active users (MAUs) in your user pool contributes important data to your planning for increases to request-rate quotas. You can compare your API request rates to the number of users you had active in a given time period. With that knowledge, you can calculate how an increase in active users of your applications will affect your quotas in your usage model. For example, imagine that your combined applications in US West (Oregon) resulted in 2 million active users in a month and your UserAuthentication category received occasional throttling errors at the default quota of 120 requests per second (RPS). In the previous month, before your successful advertising campaign, you had 1 million MAUs and your applications never exceeded 80 RPS. If you anticipate a similar spike as a result of a new TV spot, you might purchase an additional 40 RPS to accommodate the next million users with an adjusted quota of 160 RPS.

To review your MAUs

Access the [AWS Billing console](#) and review a recent bill. Under **charges by service**, you can filter on **Cognito** to view a breakdown of your MAUs for that billing period.

Requesting a quota increase

Amazon Cognito has a quota for the maximum number of operations per second that you can perform in your user pools and identity pools in each AWS Region. You can purchase an increase to adjustable Amazon Cognito user pools API request rate quotas. Check your current quota and purchase an increase from the Service Quotas console or with the Service Quotas API operations `ListAWSDefaultServiceQuotas` and `RequestServiceQuotaIncrease`.

- To purchase a quota increase using the Service Quotas console, see [Requesting a API quota increase](#) in the *Service Quotas User Guide*.
- AWS targets completion of quota increase requests within 10 days. However, several considerations might cause the request processing time to exceed 10 days. Some requests, for example, might require Amazon Cognito to provision additional hardware capacity, and seasonal increases in request volumes might introduce delays.
- If the quota isn't available in Service Quotas, use the [Service limit increase form](#).

 **Important**

Only adjustable quotas can be increased. You must purchase increased quota capacity. For quota-increase pricing, see [Amazon Cognito pricing](#).

Amazon Cognito user pools API operation categories and request rate quotas

Because Amazon Cognito has overlapping classes of API operations with [differing authorization models](#), each operation belongs to a category. Each category has its own pooled quota for all member API operations, across all user pools in one AWS Region in your account. You can only request an increase to *adjustable* category quotas. For more information, see [Requesting a quota increase](#). Quota adjustments apply to the user pools in your account in a single Region. Amazon Cognito restricts operations in some categories³ to 5 requests per second (RPS), per user pool. The **Default quota (RPS)** additionally applies to all user pools in an AWS account.

 **Note**

The quota for each category is measured in Monthly Active Users (MAUs). AWS accounts with fewer than two million MAUs can operate within the default quota. If you have less than one million MAUs and Amazon Cognito is throttling requests, consider optimizing your app. For more information, see [Optimize request rates for quota limits](#).

Category operation quotas are applied across all users in all user pools within one AWS Region. Amazon Cognito also maintains a quota for the number of requests that your app can generate against one user. You must limit per-user API requests as shown in the following table.

Amazon Cognito user pools per-user request rate quotas

Operation	Operations per user per second
Read user profile Examples: GetUser, GetDevice , InitiateAuth , RespondTo AuthChallenge	10
Write user profile Examples: UpdateUserAttributes , SetUserSettings	10

You must limit per-category API requests as shown in the following table.

Amazon Cognito user pools per-category request rate quotas

Category	Description	Default quota (RPS)	Adjustable
UserAuthentication	<p>Operations that authenticate (sign in) a user.</p> <ul style="list-style-type: none"> • InitiateAuth • Token refresh with InitiateAuth or Token endpoint • RespondToAuthChallenge¹ • AdminInitiateAuth • AdminRespondToAuthChallenge¹ • Hosted UI sign-in and MFA in 	120	Yes

Category	Description	Default quota (RPS)	Adjustable
authorization-code or implicit grants²			
UserCreation <ul style="list-style-type: none"> • SignUp • ConfirmSignUp • AdminCreateUser • AdminConfirmSignUp 	Operations that create or confirm an Amazon Cognito <i>local user</i> . This is a user that is created and verified directly by your Amazon Cognito user pools.	50	Yes
UserFederation Operations that federate (authenticate) users with a third-party identity provider into your Amazon Cognito user pools.	Operations that submit an IdP response to a user pool federation endpoint. OIDC or social provider operations that result in an IdP token, and all SAML requests, contribute to this quota.	25	Yes

Category	Description	Default quota (RPS)	Adjustable
UserAccountRecovery • ChangePassword • ConfirmForgotPassword • ForgotPassword • AdminResetUserPassword • AdminSetUserPassword • RespondToAuthChallenge¹ • AdminRespondToAuthChallenge¹ • Managed login password reset	Operations that recover a user's account, or change or update a user's password.	30	No
UserRead • Admin GetUser • GetUser	Operations that retrieve a user from your user pools.	120	Yes

Category	Description	Default quota (RPS)	Adjustable
UserUpdate <ul style="list-style-type: none"> • AdminAddUserToGroup • AdminDeleteUserAttributes • AdminUpdateUserAttributes • AdminDeleteUser • AdminDisableUser • AdminEnableUser • AdminLinkProviderForUser • AdminDisableProviderForUser • VerifyUserAttribute • DeleteUser • DeleteUserAttributes • UpdateUserAttributes • AdminUserGlobalSignOut • GlobalSignOut • AdminRemoveUserFromGroup 	Operations that you use to manage users and user attributes.	25	No
UserToken <ul style="list-style-type: none"> • RevokeToken 	Operations for token management	120	Yes

Category	Description	Default quota (RPS)	Adjustable
UserResourceRead	<p>Operations that retrieve user resource information from Amazon Cognito, such as a remembered device or a group membership.</p> <ul style="list-style-type: none">• AdminGetDevice• AdminListGroupsForUser• AdminListDevices• GetDevice• ListDevices• GetUserAttributeVerificationCode• ResendConfirmationCode• AdminListUserAuthEvents	50	Yes

Category	Description	Default quota (RPS)	Adjustable
UserResourceUpdate <ul style="list-style-type: none"> • AdminForgetDevice • AdminUpdateAuthEventFeedback • AdminSetUserMFAPreference • AdminSetUserSettings • AdminUpdateDeviceStatus • UpdateDeviceStatus • UpdateAuthEventFeedback • ConfirmDevice • SetUserMFAPreference • SetUserSettings • VerifySoftwareToken • AssociateSoftwareToken • ForgetDevice 	Operations that update resource information for a user, such as a remembered device or a group membership.	25	No
UserList <ul style="list-style-type: none"> • ListUsers • ListUsersInGroup 	Operations that return a list of users.	30	No

Category	Description	Default quota (RPS)	Adjustable
UserPoolRead <ul style="list-style-type: none">• DescribeUserPool• ListUserPools	Operations that read your user pools.	15	No
UserPoolUpdate <ul style="list-style-type: none">• CreateUserPool• UpdateUserPool• DeleteUserPool	Operations that create, update, or delete your user pools.	15	No

Category	Description	Default quota (RPS)	Adjustable
UserPoolResourceRead <ul style="list-style-type: none"> • DescribeIdentityProvider • DescribeResourceServer • DescribeUserImportJob • DescribeUserPoolDomain • GetCSVHeader • GetGroup • GetSigningCertificate • GetIdentityProviderByIdentifier • GetUserPoolMfaConfig • ListGroups • ListIdentityProviders • ListResourceServers • ListTagsForResource • ListUserImportJobs • DescribeRiskConfiguration • GetUICustomization 	Operations that retrieve information about resources, such as groups or resource servers, from a user pool. ³	20	No

Category	Description	Default quota (RPS)	Adjustable
<p>UserPoolResourceUpdate</p> <ul style="list-style-type: none"> • AddCustomAttributes • CreateGroup • CreateIdentityProvider • CreateDataSourceServer • CreateUserImportJob • CreateUserPoolDomain • DeleteGroup • DeleteIdentityProvider • DeleteDataSourceServer • DeleteUserPoolDomain • SetUserPoolMfaConfig • StartUserImportJob • StopUserImportJob • UpdateGroup • UpdateIdentityProvider • UpdateDataSourceServer 	<p>Operations that modify resources, such as groups or resource servers, in a user pool.³</p>	15	No

Category	Description	Default quota (RPS)	Adjustable
<ul style="list-style-type: none"> • UpdateUserPoolDomain • SetRiskConfiguration • SetUICustomization • TagResource • UntagResource 			
UserPoolClientRead <ul style="list-style-type: none"> • DescribeUserPoolClient • ListUserPoolClients 	Operations that retrieve information about your user pool clients. ³	15	No
UserPoolClientUpdate <ul style="list-style-type: none"> • CreateUserPoolClient • DeleteUserPoolClient • UpdateUserPoolClient 	Operations that create, update, and delete your user pool clients. ³	15	No
ClientAuthentication client_credentials grant type requests to the token endpoint.	Operations that generate credentials to be used in authorizing machine-to-machine requests	150	No

¹ A RespondToAuthChallenge or AdminRespondToAuthChallenge response with a ChallengeName of NEW_PASSWORD_REQUIRED counts toward the UserAccountRecovery category. All other challenge responses count toward the UserAuthentication category.

² Each hosted UI operation during sign-in contributes one request to the quota. For example, a user who signs in and provides an MFA code contributes 2 requests. Token redemption in authorization-code grants is subject to an additional quota allocation at the same rate as your quota in the UserAuthentication category.

³ 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.

Amazon Cognito identity pools (federated identities) API operation request rate quotas

Operation	Description	Default quota (RPS) ¹	Adjustable	Quota increase eligibility
GetId	Retrieve an identity ID from an identity pool.	25	Yes	Contact your account team.
GetOpenIdToken	Retrieve an OpenID token from an identity pool in the classic workflow.	200	Yes	Contact your account team.
GetCredentialsForIdentity	Retrieve AWS credentials from an identity pool in the enhanced workflow.	200	Yes	Contact your account team.
GetOpenIdTokenForDeveloperIdentity	Retrieve an OpenID token from an identity	50	Yes	Contact your account team.

Operation	Description	Default quota (RPS) ¹	Adjustable	Quota increase eligibility
CreateIdentityPool	pool in the developer workflow.			
ListIdentities	Retrieve a list of identity IDs in an identity pool.	5	Yes	Contact your account team.
DeleteIdentities	Delete one or more registered identities from an identity pool.	10	Yes	Contact your account team.
TagResource	Apply a tag to an identity pool.	5	Yes	Contact your account team.
UntagResource	Remove a tag from an identity pool.	5	Yes	Contact your account team.
ListTagsForResource	Display a list of the tags applied to an identity pool.	10	Yes	Contact your account team.

¹ The default quota is the minimum request rate quota for the identity pools in any AWS Region in your AWS account. Your RPS quota might be higher in some Regions.

Quotas on resource number and size

Resource quotas are the maximum number or size of resources, input fields, time duration, and other miscellaneous features in Amazon Cognito.

You can request an adjustment to some resource quotas in the Service Quotas console or from a [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 isn't available in Service Quotas, use the [Service limit increase form](#).

 **Note**

Resource quotas at the AWS account level, like *User pools per Region*, apply to Amazon Cognito resources in each AWS Region. For example, you can have 1,000 user pools in US East (N. Virginia) and another 1,000 in Europe (Stockholm).

The following tables indicate default resource quotas, and whether they're adjustable.

Amazon Cognito user pools resource quotas

Resource	Quota	Adjustable	Maximum quota
App clients per user pool	1,000	Yes	10,000
User pools per Region	1,000	Yes	10,000
Identity providers per user pool	300	Yes	1,000
Resource servers per user pool	25	Yes	300
Users per user pool	40,000,000	Yes	Contact your account team.
Total combined changes in pre token generation Lambda trigger ¹	5,000	Yes	Contact your account team.
Custom attributes per user pool	50	No	N/A

Resource	Quota	Adjustable	Maximum quota
Characters per attribute	2,048 bytes	No	N/A
Characters in custom attribute name	20	No	N/A
Required minimum password characters in password policy	6–99	No	N/A
Email messages sent daily per AWS account ²	50	No	N/A
Characters in email subject	140	No	N/A
Characters in email message	20,000	No	N/A
Characters in SMS verification message	140	No	N/A
Characters in password	256	No	N/A
Characters in identity provider name	32	No	N/A
Characters in a SAML response	100,000	No	N/A
Identifiers per identity provider	50	No	N/A
Identities linked to a user	5	No	N/A

Resource	Quota	Adjustable	Maximum quota
Callback URLs per app client	100	No	N/A
Logout URLs per app client	100	No	N/A
Scopes per resource server	100	No	N/A
Scopes per app client	50	No	N/A
Custom domains per account	4	No	N/A
Groups to which each user can belong	100	No	N/A
Groups per user pool	10,000	No	N/A

¹ This quota might be encountered in tokens from a [Pre token generation Lambda trigger](#). The number of existing and added claims plus scopes in access and identity tokens in one transaction must add up to a number smaller than or equal to this quota. Suppressed claims and scopes don't contribute to this quota.

² This quota applies only if you are using the default email feature for an Amazon Cognito user pool. For 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](#).

Amazon Cognito user pools session validity parameters

Token	Quota
ID token	5 minutes – 1 day
Refresh token	1 hour – 3,650 days
Access token	5 minutes – 1 day

Token	Quota
Hosted UI session cookie	1 hour
Authentication session token	3 minutes – 15 minutes

Amazon Cognito user pools code security resource quotas (non-adjustable)

Resource	Quota
Sign-up confirmation code validity period	24 hours
User attribute verification code validity period	24 hours
Multi-factor authentication (MFA) code validity period	3–15 minutes
Forgot password code validity period	1 hour
Maximum number of ConfirmForgotPassword and ForgotPassword requests per user per hour ¹	5–20
Maximum number of ResendConfirmationCode requests per user per hour	5
Maximum number of ConfirmSignUp requests per user per hour	15
Maximum number of ChangePassword requests per user per hour	5
Maximum number of GetUserAttributeVerificationCode requests per user per hour	5
Maximum number of VerifyUserAttribute requests per user per hour	15

¹ Amazon Cognito evaluates risk factors in the request to update passwords and assigns a quota that's tied to the evaluated risk level. For more information, see [Forgot password behavior](#).

Amazon Cognito user pools user import job resource quotas

Resource	Quota	Adjustable	Maximum quota
User import jobs per user pool	1,000	Yes	Contact your account team.
Maximum characters per user import CSV row	16,000	No	N/A
Maximum CSV file size	100 MB	No	N/A
Maximum number of users per CSV file	500,000	No	N/A

Amazon Cognito identity pools (federated identities) resource quotas

Resource	Quota	Adjustable	Maximum quota
Identity pools per account	1,000	Yes	N/A
Amazon Cognito user pool providers per identity pool	50	Yes	1000
Character length of an identity pool name	128 bytes	No	N/A
Character length of a login provider name	2,048 bytes	No	N/A

Resource	Quota	Adjustable	Maximum quota
Identities per identity pool	Unlimited	No	N/A
Identity providers for which role mappings can be specified	10	No	N/A
Results from a single list or lookup call	60	No	N/A
Role-based access control (RBAC) rules	25	No	N/A

Amazon Cognito Sync resource quotas

Resource	Quota	Adjustable	Maximum quota
Datasets per identity	20	Yes	Contact your account team.
Records per dataset	1,024	Yes	Contact your account team.
Size of a single dataset	1 MB	Yes	Contact your account team.
Characters in dataset name	128 bytes	No	N/A
Waiting time for a bulk publish after a successful request	24 hours	No	N/A

Document history for Amazon Cognito

The following table describes important additions to the documentation for Amazon Cognito. We also make frequent minor updates to the documentation in response to the feedback that you send. To submit feedback, locate the **Feedback** link at the bottom of any page in Amazon Cognito documentation.

Change	Description	Date
<u>Updates to getting-started resources for user pools.</u>	The getting started experience with Amazon Cognito user pools has a new console design and application options.	November 21, 2024
<u>New pricing model with feature plans.</u>	Updated the billing model for user pools. Advanced security features are now threat protection. Components in the advanced security features license are now in the Essentials and Plus feature plans.	November 21, 2024
<u>New managed login feature.</u>	Launched managed login, an update to the hosted UI.	November 21, 2024
<u>A new authentication method and new authentication flows.</u>	You can now sign in to Amazon Cognito user pools with passkeys and one-time passwords.	November 21, 2024
<u>Added login_hint parameter.</u>	You can now add a username hint to authorization requests for the hosted UI, OIDC IdPs, and Google IdPs.	October 3, 2024

<u>New advanced security features for email MFA.</u>	You can now send multi-factor authentication (MFA) codes by email message with advanced security features.	September 12, 2024
<u>New content and page changes.</u>	Modified titles, removed unneeded content, added scenario-based intros, moved user pools OIDC & hosted UI endpoints reference to user pools section.	September 9, 2024
<u>Updated information about AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy .</u>	The AWS managed policy for scope-down of unauthenticated identities in identity pools now permits Amazon Location Service.	August 9, 2024
<u>New threat prevention for custom authentication with Lambda triggers and enhanced threat detection.</u>	You can now analyze custom authentication sign-in with threat protection and apply adaptive authentication responses. Threat protection also now analyzes sign-in traffic for impossible geographical distance between attempts.	August 8, 2024
<u>New advanced security features for password reuse prevention and user-activity log export.</u>	You can now export user activity logs and set a password-history policy with advanced security features in Amazon Cognito user pools.	August 6, 2024

<u>Amazon Cognito is now available in the Canada West (Calgary) and Asia Pacific (Hong Kong) AWS Regions.</u>	You can now create Amazon Cognito resources in the Canada West (Calgary) and Asia Pacific (Hong Kong) Regions.	July 9, 2024
<u>Improved description of application behavior for advanced security</u>	Updated information about device context data for advanced security adaptive authentication.	June 10, 2024
<u>Added support for complex objects in pre token Lambda trigger</u>	You can now add arrays and JSON objects to ID and access token claims.	May 30, 2024
<u>Updated information about Verified Permissions and Amazon Cognito.</u>	Amazon Verified Permissions now has more direct integration with Amazon Cognito.	May 15, 2024
<u>Multi-Region Amazon SES verified identities.</u>	In some AWS Regions without Amazon SES, Amazon Cognito user pools load balance email between two remote Regions.	May 10, 2024
<u>Added information about M2M authorization and managing costs.</u>	Learn how to use client credentials grants for machine-to-machine (M2M) use cases with Amazon Cognito user pools.	May 9, 2024
<u>Amazon Cognito is now available in the Europe (Spain) and Asia Pacific (Hyderabad) AWS Regions.</u>	You can now create Amazon Cognito resources in the Europe (Spain) and Asia Pacific (Hyderabad) Regions.	April 15, 2024
<u>Amazon Cognito is now available in the Asia Pacific (Melbourne) AWS Region.</u>	You can now create Amazon Cognito resources in the Asia Pacific (Melbourne) Region.	April 4, 2024

<u>Added an example Android app in Flutter for Amazon Cognito user pools.</u>	You can build a starter mobile app for Amazon Cognito from an example Flutter application on GitHub.	April 4, 2024
<u>New getting-started content</u>	Expanded content for getting started, common scenarios, multi-tenant best practices, and accessing resources after sign-in.	April 1, 2024
<u>Amazon Cognito is now available in the Europe (Zurich) AWS Region.</u>	You can now create Amazon Cognito resources in the Europe (Zurich) Region.	March 14, 2024
<u>Amazon Cognito is now available in the Middle East (UAE) AWS Region.</u>	You can now create Amazon Cognito resources in the Middle East (UAE) Region.	March 8, 2024
<u>New SAML features and improved content.</u>	You can now sign SAML requests, encrypt SAML responses, and set up IdP-initiated SAML SSO.	February 1, 2024
<u>Quota increases available.</u>	You can now purchase additional capacity for Amazon Cognito request-rate quotas.	January 25, 2024
<u>Amazon Cognito identity pools support request rates in Service Quotas.</u>	You can now monitor requests-per-second (RPS) quotas for Amazon Cognito identity pools and request increase in the Service Quotas console.	December 19, 2023

<u>Added a new feature for customization of the contents of access tokens.</u>	You can now add, modify, and remove claims and scopes in user pool access tokens.	December 12, 2023
<u>Improved content about app clients and OAuth scopes.</u>	Clarity edits and corrections to Application-specific settings with app clients and Scopes, M2M, and APIs with resource servers . Removed legacy console instructions.	November 14, 2023
<u>Improved content about devices and device authentication.</u>	New content about the use of device keys and device SRP authentication.	October 18, 2023
<u>Updated AWS Management Console guidance.</u>	Removed user pools console reference and redistributed topics within related subjects, and added guidance to tab-based organization in Amazon Cognito console.	August 30, 2023
<u>De-emphasized direct access to LOGIN endpoint.</u>	Added a visual overview of the user pool Login endpoint and emphasized starting authentication with Authorize endpoint .	August 30, 2023
<u>Amazon Cognito is now available in the Asia Pacific (Osaka) and Israel (Tel Aviv) AWS Regions.</u>	You can now create Amazon Cognito resources in the Asia Pacific (Osaka) and Israel (Tel Aviv) Regions.	August 30, 2023
<u>Introduced information about authorization for Amazon Cognito with Amazon Verified Permissions.</u>	In your app, you can invoke the Verified Permissions API to produce access decisions from a central authority.	August 1, 2023

<u>Added a new feature for logging user pool detailed user activity to Amazon CloudWatch Logs.</u>	You can now log email and SMS message delivery errors to CloudWatch log groups.	August 1, 2023
<u>Updated information about AWS managed policy for identity pool guest users.</u>	Permissions scope-down for identity pool guest users now includes both an inline session policy and an AWS managed session policy.	May 16, 2023
<u>Content improvement and new console instructions for Amazon Cognito identity pools.</u>	Added new console walkthroughs to reflect the new console experience, improved code integration details for identity pools.	May 16, 2023
<u>Additions and improvements to service homepage and user pools homepage.</u>	Updated overview pages for Amazon Cognito and <u>user pools</u> .	May 16, 2023
<u>General improvements to user pool token documentation.</u>	Updated example tokens, added new information about verifying tokens.	February 16, 2023
<u>You can now log Amazon Cognito identity pools data events in AWS CloudTrail.</u>	CloudTrail supports the selection of Amazon Cognito identity pools high-volume API operations in trails that log data events.	February 15, 2023
<u>Updated Lambda trigger examples and descriptions.</u>	Lambda trigger examples are updated to JavaScript version 3. You can now directly correlate Lambda triggers to API actions.	January 31, 2023

<u>Amazon Cognito identity pools apply an AWS managed policy to unauthenticated sessions.</u>	Identity pool users who authenticate using the enhanced flow now have an additional AWS managed policy applied to their session.	January 31, 2023
<u>Added code examples.</u>	This guide now includes example code for your Amazon Cognito app in a variety of programming langages.	January 23, 2023
<u>Added information about API models and authentication with Amazon Cognito user pools.</u>	Amazon Cognito user pools have multiple API interfaces and formats for request authorization.	December 15, 2022
<u>Amazon Cognito is now available in the Europe (Milan) AWS Region.</u>	You can now create Amazon Cognito user pools in the Europe (Milan) Region.	December 6, 2022
<u>Added information about user pool deletion protection.</u>	When you create a new user pool with the AWS Management Console, it's now protected against deletion by default.	October 20, 2022
<u>Added a user guide for the hosted UI, and information about TOTP MFA in the hosted UI.</u>	Your users can now register a TOTP MFA device in the Amazon Cognito hosted UI. You can now preview the default hosted UI.	September 8, 2022
<u>Added information about AWS WAF and Amazon Cognito.</u>	You can now associate a AWS WAF web ACL with a Amazon Cognito user pool.	August 3, 2022

<u>Added more example AWS CloudTrail events.</u>	Amazon Cognito now logs federation and hosted UI requests to your trail.	June 15, 2022
<u>Added information about two-step attribute verification.</u>	You can now choose whether your user must verify a new email address or phone number before they can sign in with it.	June 9, 2022
<u>Updated federation documentation. New IP address propagation feature.</u>	Updated walkthroughs for setting up user pool social IdPs. Added information about federated user profiles and attribute mapping. Added new information about device fingerprints for advanced security.	May 31, 2022
<u>Sign in federated users without interaction with the hosted UI</u>	Added a new page about how to bookmark applications so that Amazon Cognito silently directs users to federated sign-in.	May 29, 2022
<u>In-Region SMS and email messaging for Amazon Cognito user pools</u>	You can now use Amazon Simple Notification Service for SMS messages and Amazon Simple Email Service for email messages in the same AWS Region as your user pool.	March 14, 2022
<u>Updates to quotas page</u>	Added and clarified resource and request-rate quotas.	January 10, 2022

<u>New Amazon Cognito user pools console experience</u>	Updated instructions to create and manage user pools in the updated Amazon Cognito console.	November 18, 2021
<u>RevokeToken API and Revocation Endpoint</u>	You can use the RevokeToken operation to revoke a refresh token for a user.	June 10, 2021
<u>Multi-tenant best practices</u>	Added best practices for multi-tenant applications.	March 4, 2021
<u>Attributes for access control</u>	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
<u>Custom SMS Sender Lambda Trigger and Custom Email Sender Lambda Trigger</u>	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 30, 2020
<u>Amazon Cognito token updates</u>	Updated expiration information was added to Access, ID, and Refresh tokens.	October 29, 2020

<u>Amazon Cognito Service Quotas</u>	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
<u>Amazon Cognito quota categorization</u>	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
<u>Amazon Cognito supported in US AWS GovCloud</u>	Amazon Cognito is now supported in the AWS GovCloud (US) Region.	May 13, 2020
<u>Amazon Cognito Pinpoint document updates</u>	New service-linked role was added. Instructions were updated on "Using Amazon Pinpoint Analytics with Amazon Cognito User Pools".	May 13, 2020

<u>New Amazon Cognito dedicated security chapter</u>	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
<u>Amazon Cognito Identity Pools now supports Sign in with Apple</u>	Sign in with Apple is available in all regions where Amazon Cognito operates, except cn-north-1 region.	April 7, 2020
<u>New Facebook API Versioning</u>	Added version selection to Facebook API.	April 3, 2020
<u>Username case insensitivity update</u>	Added recommendation about enabling username case insensitivity before creating a user pool.	February 11, 2020
<u>New information about AWS Amplify</u>	Added information about integrating Amazon Cognito 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

<u>New attribute for user pool triggers</u>	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
<u>Updated limit</u>	The throttling limit for the ListUsers API action is updated.	June 25, 2019
<u>New limit</u>	The soft limits for user pools now include a limit for the number of users.	June 17, 2019
<u>Amazon SES email settings for Amazon Cognito user pools</u>	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.	April 8, 2019
<u>Tagging support</u>	Added information about tagging Amazon Cognito resources.	March 26, 2019

<u>Change the certificate for a custom domain</u>	If you use a custom domain to host the Amazon Cognito hosted UI, you can change the SSL certificate for this domain as needed.	December 19, 2018
<u>New limit</u>	A new limit is added for the maximum number of groups that each user can belong to.	December 14, 2018
<u>Updated limits</u>	The soft limits for user pools are updated.	December 11, 2018
<u>Documentation update for verifying email addresses and phone numbers</u>	Added information about configuring your user pool to require email or phone verification when a user signs up in your app.	November 20, 2018
<u>Documentation update for testing emails</u>	Added guidance for initiating emails from Amazon Cognito while you test your app.	November 13, 2018
<u>Amazon Cognito Advanced Security</u>	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
<u>Custom Domains for Amazon Cognito Hosted UI</u>	Allow developers to use their own fully custom domain for the hosted UI in Amazon Cognito User Pools.	June 4, 2018

<u>Amazon Cognito User Pools</u>	Added user pool sign-in through an OpenID Connect (OIDC) identity provider such as Salesforce or Ping Identity.	May 17, 2018
<u>OIDC Identity Provider</u>		
<u>Amazon Cognito Lambda Migration Trigger</u>	Added pages covering the Lambda Migration Trigger feature	April 8, 2018
<u>Amazon Cognito Developer Guide Update</u>	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
<u>Amazon Cognito Advanced Security Beta</u>	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
<u>Amazon Pinpoint integration</u>	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.	September 26, 2017

<u>Federation and built-in app UI features of Amazon Cognito user pools</u>	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
<u>HIPAA and PCI compliance-related feature changes</u>	Added the ability to allow your users to use a phone number or email address as their user name.	July 6, 2017
<u>User groups and role-based access control features</u>	Added administrative capability to create and manage user groups. Administrators can assign IAM roles to users based on group membership and administrator-created rules.	December 15, 2016
<u>Documentation update</u>	Updated examples that show how to use AWS Lambda triggers with user pools.	November 27, 2016
<u>Documentation update</u>	Updated iOS code examples.	November 18, 2016
<u>Documentation update</u>	Added information about confirmation flow for user accounts.	November 9, 2016
<u>Create user accounts feature</u>	Added administrative capability to create user accounts through the Amazon Cognito console and the API.	October 6, 2016

<u>User import feature</u>	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.	September 1, 2016
<u>General availability of Cognito User Pools</u>	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.	July 28, 2016
<u>SAML support</u>	Added support for authentication with identity providers through Security Assertion Markup Language 2.0 (SAML 2.0).	June 23, 2016
<u>CloudTrail integration</u>	Added integration with AWS CloudTrail.	February 18, 2016
<u>Integration of events with Lambda</u>	Enables you to execute an AWS Lambda function in response to important events in Amazon Cognito.	April 9, 2015
<u>Data stream to Amazon Kinesis</u>	Provides control and insight into your data streams.	March 4, 2015
<u>OpenID Connect support</u>	Enables support for OpenID Connect providers.	November 23, 2014
<u>Push synchronization</u>	Enables support for silent push synchronization.	November 6, 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.

September 29, 2014

[Amazon Cognito general availability](#)

July 10, 2014