



デベロッパーガイド

Amazon Cognito



Amazon Cognito: デベロッパーガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスと関連付けてはならず、お客様に混乱を招くような形や Amazon の信用を傷つけたり失わせたりする形で使用することはできません。Amazon が所有しない商標はすべてそれぞれの所有者に所属します。所有者は必ずしも Amazon と提携していたり、関連しているわけではありません。また、Amazon 後援を受けているとはかぎりません。

Table of Contents

Amazon Cognito とは	1
ユーザープール	2
アイデンティティプール	3
Amazon Cognito の機能	4
ユーザープール	4
アイデンティティプール	7
Amazon Cognito ユーザープールとアイデンティティプールの比較	9
Amazon Cognito の使用開始方法	13
リージョナルな可用性	13
Amazon Cognito の料金	13
用語と概念	14
全般	14
ユーザープール	17
アイデンティティプール	21
の開始方法 AWS	22
にサインアップする AWS アカウント	23
管理アクセスを持つユーザーを作成する	23
ユーザープールの開始方法	26
最初のアプリケーションとユーザープール	26
その他のアプリケーションオプション	29
React SPA のサンプル	30
Flutter モバイルアプリケーションのサンプル	34
次の手順	38
ソーシャルプロバイダーの追加	39
SAML IdP を追加する	47
ID プールの使用開始方法	50
Amazon Cognito で ID プールを作成する	50
SDK の設定	52
ID プロバイダーを統合する	53
認証情報を取得する	53
サンプルアプリケーション	53
前提条件	55
認証プロバイダーの設定	56
デモアプリケーションをデプロイする	56

アイデンティティプールの認証方法を調べる	58
次の手順	88
その他の入門オプション	89
アプリケーションとの統合	91
による認証 AWS Amplify	92
Amplify によるユーザーインターフェイス (UI) の作成	93
AWS SDKsによる認証	94
認証の仕組み	95
マネージドログイン認証	96
SDK 認証	99
サードパーティー ID プロバイダー認証	102
アイデンティティプール認証	105
AWS SDKs の使用	108
Amazon Verified Permissions による承認	109
Verified Permissions を使用した API 認可	111
Amazon Cognito ユーザーのポリシーの例。	114
コードの例	117
Amazon Cognito ID	119
基本	119
シナリオ	145
Amazon Cognito ID プロバイダー	147
基本	148
シナリオ	313
Amazon Cognito Sync	480
基本	481
マルチテナンシーのベストプラクティス	483
テナントごとのユーザープール	485
テナントごとのアプリケーションクライアント	487
テナントごとのユーザープールグループ	489
テナントごとのカスタム属性	491
テナントごとのカスタムスコープ	493
リソースの例	496
マルチテナンシーのセキュリティに関する推奨事項	497
一般的な Amazon Cognito シナリオ	499
ユーザープールを使用して認証する	499
サーバー側のリソースへのアクセス	500

API Gateway と Lambda を使用してリソースにアクセスする	501
ユーザープールと ID プールを使用して AWS サービスにアクセスする	502
サードパーティーによる認証と ID プールによる AWS サービスへのアクセス	503
Amazon Cognito で AWS AppSync リソースにアクセスする	504
Amazon Cognito user pools	506
機能	507
サインアップ	507
サインイン	508
マネージドログイン	509
セキュリティ	509
カスタマーユーザーエクスペリエンス	510
モニタリングと分析	510
Amazon Cognito アイデンティティプール統合	511
ユーザープールの機能プラン	511
機能プランを選択する	514
プラン別の機能	515
エッセンシャルプランの機能	518
プラスプランの機能	522
対象外の機能をオフにする	525
セキュリティのベストプラクティス	526
ネットワークレベルでユーザープールを保護する	526
SMS メッセージの不正使用から保護する	527
パブリック認証を理解する	527
秘密クライアントをクライアントシークレットで保護する	531
その他のシークレットを保護する	531
ユーザープール管理の最小特権	532
トークンの保護と検証	535
信頼する ID プロバイダーを決定する	535
ユーザープロフィールへのアクセスに対するスコープの影響を理解する	536
ユーザー属性の入力をサニタイズする	536
認証	537
認証フローを実装する	538
主要事項	540
認証フローの例	543
マネージドログイン認証	546
SDK 認証	549

認証フロー	554
SDK 認可モデル	580
サードパーティの IdP によるサインイン	596
Amazon Cognito ユーザープールでのフェデレーションサインインの仕組み	596
Amazon Cognito のサービスプロバイダーとしてのアプリの責任	598
Amazon Cognito ユーザープールサードパーティーサインインについて知っておくべきこと	598
ID プロバイダ	600
ソーシャル ID プロバイダー	606
SAML プロバイダー	616
OIDC プロバイダー	649
IdP 属性をマッピングする	660
フェデレーションユーザーをリンクする	667
マネージドログイン	671
マネージドログインのローカリゼーション	673
規約ドキュメント	674
を使用したマネージドログインの設定 AWS Amplify	676
Amazon Cognito コンソールでのマネージドログインの設定	676
サインインページの表示	677
認証ページのカスタマイズ	678
マネージドログインとホストされた UI について知っておくべきこと	679
ドメインの設定	682
ブランディングとカスタマイズ	696
Lambda トリガーの使用	717
主要事項	720
トリガーを設定する	722
ユーザープールの Lambda トリガーイベント	723
ユーザープールの Lambda トリガーの一般的なパラメータ	724
クライアントメタデータ	725
オペレーション別のトリガーソース	728
機能別のトリガーソース	735
サインアップ前	740
確認後	748
事前認証	751
認証後	755
インバウンドフェデレーション	759

カスタムチャレンジ	769
トークン生成前	791
ユーザー移行	815
カスタムメッセージ	821
カスタム送信者	829
ユーザーの管理	845
ユーザーのサインアップの許可	846
ユーザーアカウントのサインアップと確認	849
管理者としてのユーザーの作成	877
ユーザープールにグループを追加する	884
ユーザーの管理と検索	888
パスワード	896
ユーザープールへのユーザーのインポート	902
属性	921
ユーザープールトークン	938
ID トークン	940
アクセストークン	944
更新トークン	949
トークンの取り消し	955
JSON Web トークンの検証	957
キャッシュトークン	964
サインイン後にリソースにアクセスする	968
Verified Permissions を使用したリソースへのアクセス	500
API Gateway リソースへのアクセス	971
ID プールを使用した AWS リソースへのアクセス	972
M2M とスコープ	977
API 認可	978
Machine to machine (M2M) 認可	978
スコープについて	980
リソースサーバーについて	982
リソースバインディング	986
その他の機能	987
ユーザープールとアプリケーションクライアントの更新	988
アプリクライアント	993
「」 デバイスの使用	1003
Amazon Pinpoint 分析の使用	1009

E メール設定	1015
SMS メッセージ設定	1030
セキュリティ機能を使用する	1040
MFA の追加	1042
脅威保護	1064
AWS WAF ウェブ ACLs	1093
大文字と小文字の区別	1098
削除保護	1100
ユーザー開示の管理	1101
ユーザープールのエンドポイントリファレンス	1108
マネージドログインのエンドポイント	1109
フェデレーションエンドポイント	1117
OAuth 2.0 許可	1151
PKCE の使用	1153
マネージドログインとフェデレーションのエラーレスポンス	1155
Amazon Cognito アイデンティティプール	1158
アイデンティティプールの設定	1160
アイデンティティプールを作成する	1161
ユーザー IAM ロール	1162
認証された ID と認証されていない ID	1163
ゲストアクセスをアクティブ化/非アクティブ化する	1163
ID の種類に関連付けられたロールを変更する	1164
ID プロバイダーを編集する	1165
アイデンティティプールを削除する	1167
ID プールから ID を削除する	1167
Amazon Cognito Sync を ID プールで使用する	1168
ID プールの認証フロー	1170
拡張 (簡易) 認証フロー	1171
基本 (クラシック) 認証フロー	1172
デベロッパーが認証した認証フロー	1174
どの認証フローを実装する必要があるか?	1176
認証フローの API オペレーションの概要	1177
IAM ロール	1181
信頼ポリシーのセットアップ	1182
アクセスポリシー	1187
ロールの信頼とアクセス権限	1199

セキュリティのベストプラクティス	1201
IAM 設定のベストプラクティス	1201
アイデンティティプール設定のベストプラクティス	1203
アクセスコントロールへの属性の使用	1205
Amazon Cognito の ID プールを用いたアクセスコントロールへの属性の使用	1206
アクセスコントロールポリシーへの属性の使用例	1208
アクセスコントロールの属性をオフにする	1210
デフォルトのプロバイダーマッピング	1210
ロールベースアクセスコントロールの使用	1213
ロールマッピング用のロールの作成	1213
pass-role 許可の付与	1214
ユーザーにロールを割り当てるためのトークンの使用	1215
ルールベースのマッピングを使用してユーザーにロールを割り当てる	1216
ルールベースのマッピングで使用するトークンクレーム	1218
ロールベースのアクセスコントロールのベストプラクティス	1219
認証情報の取得	1220
認証情報の使用	1227
サードパーティー ID プロバイダー	1229
Facebook	1230
Login with Amazon	1239
Google	1242
Apple でのサインイン	1252
Open ID Connect プロバイダー	1258
SAML ID プロバイダー	1261
デベロッパーが認証した ID	1264
認証のフローについて	1264
デベロッパー名を指定して ID プールに関連付ける	1265
ID プロバイダーの実装	1265
ログインマップの更新 (Android および iOS のみ)	1274
トークンの取得 (サーバー)	1274
既存のソーシャル ID への接続	1276
プロバイダー間の移行のサポート	1276
ID の切り替え	1281
Android	1281
iOS - Objective-C	1281
iOS - Swift	1282

JavaScript	1282
Unity	1283
Xamarin	1284
Amazon Cognito Sync	1285
Amazon Cognito Sync の使用開始方法	1286
Amazon Cognito で ID プールをセットアップする	1286
データを保存し、同期する	1286
クライアント間でのデータの同期	1286
Amazon Cognito Sync クライアントの初期化	1287
データセットについて	1289
データセットのデータの読み取りと書き込み	1291
同期ストアでのローカルデータの同期	1293
イベントコールバックの処理	1297
Android	1297
iOS - Objective-C	1299
iOS - Swift	1303
JavaScript	1306
Unity	1309
Xamarin	1312
プッシュ同期の実装	1314
Amazon Simple Notification Service (Amazon SNS) アプリケーションを作成する	1315
Amazon Cognito コンソールでプッシュ同期を有効にする	1315
アプリケーションでのプッシュ同期の使用: Android	1316
アプリケーションでプッシュ同期を使用する: iOS - Objective-C	1318
アプリケーションでプッシュ同期を使用する: iOS - Swift	1321
Amazon Cognito Sync ストリームの実装	1324
Amazon Cognito イベントによるワークフローのカスタマイズ	1327
セキュリティ	1332
データ保護	1333
データ暗号化	1333
ID とアクセス管理	1335
オーディエンス	1335
アイデンティティを使用した認証	1336
ポリシーを使用したアクセスの管理	1337
Amazon Cognito で IAM が機能する仕組み	1339
アイデンティティベースのポリシーの例	1347

トラブルシューティング	1352
サービスにリンクされたロールの使用	1354
ログ記録とモニタリング	1359
コストのモニタリング	1360
ユーザープールログのエクスポート	1363
クォータ値と使用状況のモニタリング	1374
CloudTrail ログ	1394
AWS PrivateLink	1423
AWS PrivateLink 統合の認証フロー	1424
のオペレーションモード AWS PrivateLink	1425
考慮事項	1425
リソースコントロールポリシーによるアクセスの制御	1431
インターフェイスエンドポイントの作成	1432
エンドポイントポリシーを作成する	1432
アイデンティティベースのポリシーを作成する	1434
コンプライアンス検証	1436
耐障害性	1437
リージョンデータに関する考慮事項	1437
インフラストラクチャセキュリティ	1438
設定と脆弱性の分析	1439
AWS マネージドポリシー	1439
ポリシーの更新	1441
トラブルシューティング	1444
カスタムドメインの設定エラー	1444
Custom domain is not a valid subdomain	1444
Domain already associated with another user pool	1445
One or more of the CNAMEs that you provided are already associated with a different resource	1445
The specified SSL certificate doesn't exist	1446
Invalid refresh token エラー	1446
フェデレーションでの無効な SAML レスポンスエラー	1447
Invalid user attributes: Required attribute	1448
Invalid SAML response received: SAML Response signature is invalid	1448
Audience restriction、または Application with identifier not found... ..	1449
An error was encountered with the requested page	1449
Invalid relayState from identity provider	1449

マネージドログインユーザーが MFA 要素を選択できない	1450
パスワードなしのユーザーとパスキーユーザーは MFA を使用できません	1451
E メール/SMS 経由でパスワードリセットコードを受信できない	1451
パスワードのリセットは、未検証の復旧属性で失敗します。Could not reset password for the account, please contact support or try again	1452
SECRET_HASH エラー	1453
Amazon Cognito コンソールが、新しいユーザープールにデフォルト設定を選択する	1453
その他のトラブルシューティングリソース	1454
リソースのタグ付け	1455
サポートされているリソース	1455
タグの制限	1456
コンソールを使用したタグの管理	1456
AWS CLI 例	1457
タグの割り当て	1457
タグの表示	1458
タグの削除	1458
リソース作成時のタグの適用	1459
API アクション	1460
ユーザープールタグの API アクション	1460
ID プールタグの API アクション	1460
クォータ	1462
API リクエストレートクォータについて	1462
クォータのカテゴリ	1462
リクエストレートの特別な取り扱いを使用した Amazon Cognito ユーザープール API オペ レーション	1463
月次のアクティブユーザー	1464
API リクエストレートクォータ	1466
クォータ要件を特定する	1466
リクエストレートを最適化する	1467
クォータの使用状況を追跡する	1468
月間アクティブユーザー数 (MAU) を追跡する	1469
クォータ引き上げのリクエスト	1469
ユーザープールのリクエストレートクォータ	1470
ユーザープールドメインの一括リクエストレート制限	1482
ID プールリクエストレートクォータ	1483
リソースの番号とサイズのクォータ	1485

Amazon Cognito ユーザープールのリソースクォータ	1485
Amazon Cognito ユーザープールセッションの有効性パラメータ	1489
Amazon Cognito ユーザープールコードセキュリティリソースクォータ (調整不可)	1489
Amazon Cognito ユーザープールへのユーザーインポートジョブのリソースクォータ	1490
Amazon Cognito アイデンティティプール (フェデレーテッドアイデンティティ) のリソースクォータ	1491
Amazon Cognito Sync リソースクォータ	1492
ドキュメント履歴	1494
.....	mdxviii

Amazon Cognito とは

Amazon Cognito はウェブアプリとモバイルアプリ用のアイデンティティプラットフォームです。これは、OAuth 2.0 アクセストークンと AWS 認証情報のための、ユーザーディレクトリであり、認証サーバーであり、認可サービスです。Amazon Cognito を使用すると、組み込みのユーザーディレクトリ、エンタープライズディレクトリ、Google や Facebook などのコンシューマー ID プロバイダーからユーザーを認証および認可できます。

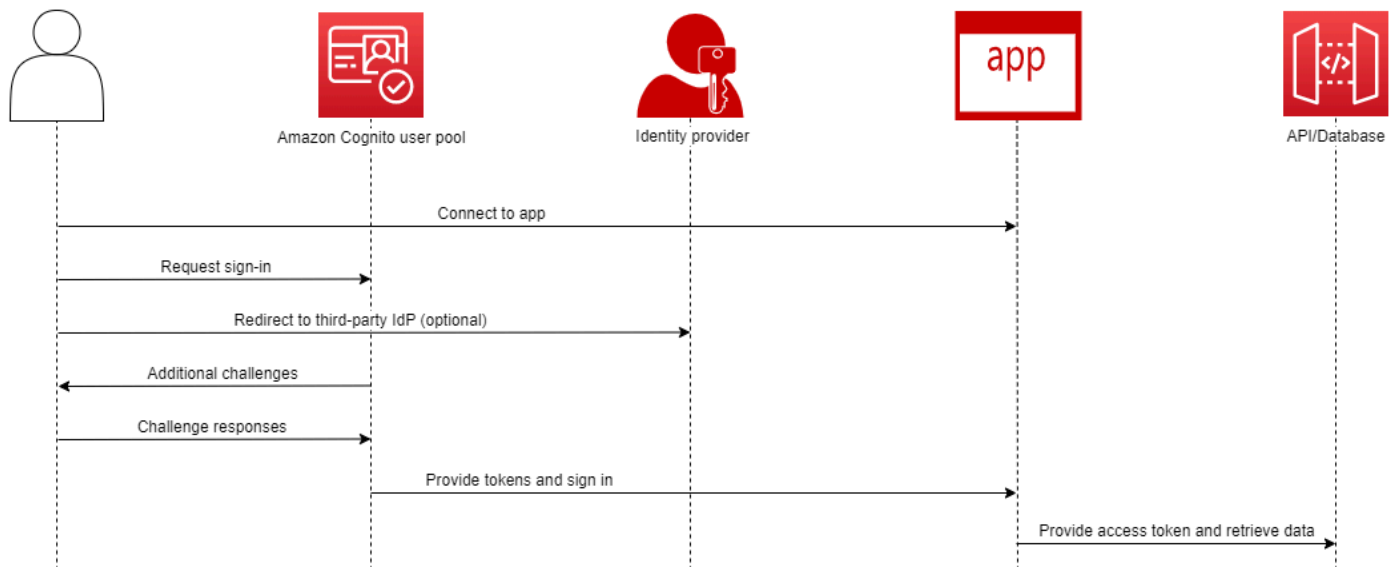
トピック

- [ユーザープール](#)
- [アイデンティティプール](#)
- [Amazon Cognito の機能](#)
- [Amazon Cognito ユーザープールとアイデンティティプールの比較](#)
- [Amazon Cognito の使用開始方法](#)
- [リージョナルな可用性](#)
- [Amazon Cognito の料金](#)
- [Amazon Cognito の一般的な用語と概念](#)
- [の開始方法 AWS](#)

その後に続く 2 つのコンポーネントが Amazon Cognito を構成します。これらは、ユーザーのアクセスニーズに応じて、独立して動作することも、連携して動作することもできます。

ユーザープール

Amazon Cognito user pools

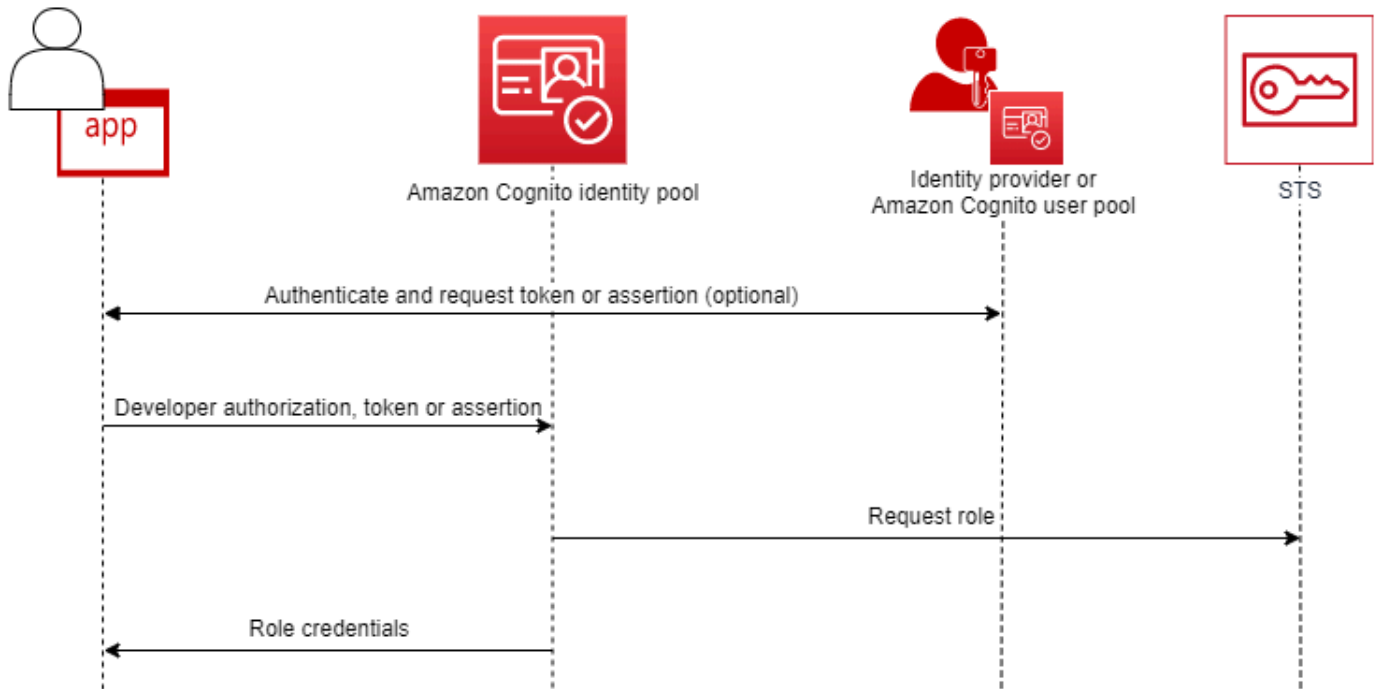


アプリまたは API のユーザーを認証および認可する場合は、ユーザープールを作成します。ユーザープールは、セルフサービスと管理者主導の両方によるユーザー作成、管理、認証を行うユーザーディレクトリです。ユーザープールは、独立したディレクトリや OIDC ID プロバイダー (IdP) でも、従業員や顧客のアイデンティティを提供するサードパーティプロバイダーの中間サービスプロバイダー (SP) でもかまいません。ユーザープールの SAML 2.0 および OIDC IdP における組織のワークフォースアイデンティティに対して、アプリでシングルサインオン (SSO) を提供できます。また、パブリック OAuth 2.0 アイデンティティストア (Amazon、Google、Apple、および Facebook) 内の組織のカスタマーアイデンティティに対して、アプリで SSO を提供することもできます。カスタマー ID およびアクセス管理 (CIAM) の詳細については、「[CIAM とは何ですか?](#)」を参照してください。

ユーザープールはアイデンティティプールとの統合を必要としません。ユーザープールから、認証された JSON ウェブトークン (JWT) をアプリ、ウェブサーバー、または API に直接発行できます。

アイデンティティプール

Amazon Cognito federated identities (identity pools)



認証されたユーザーまたは匿名ユーザーに AWS リソースへのアクセスを認可する場合は、Amazon Cognito アイデンティティプールを設定します。ID プールは、ユーザーにリソースを提供するためにアプリの AWS 認証情報を発行します。ユーザープールや SAML 2.0 サービスなど、信頼できる ID プロバイダーでユーザーを認証できます。また、オプションでゲストユーザーに認証情報を発行することもできます。ID プールは、ロールベースのアクセスコントロールと属性ベースのアクセスコントロールの両方を使用して、ユーザーの AWS リソースへのアクセス許可を管理します。

アイデンティティプールはユーザープールとの統合を必要としません。アイデンティティプールは、ワークフォースとコンシューマーの両方の ID プロバイダーからの認証済みクレームを直接受け入れることができます。

Amazon Cognito のユーザープールとアイデンティティプールの併用

このトピックの最初の図では、Amazon Cognito を使用してユーザーを認証し、ユーザーにアクセス権を付与します。AWS のサービス

1. アプリユーザーはユーザープールを介してサインインし、OAuth 2.0 トークンを受け取ります。
2. アプリは、AWS APIs および AWS Command Line Interface () で使用できる一時的な AWS 認証情報として、ユーザープールトークンを ID プールと交換しますAWS CLI。
3. アプリは認証情報セッションをユーザーに割り当て、Amazon S3 や Amazon DynamoDB AWS のサービスなどの に承認されたアクセスを提供します。

アイデンティティプールとユーザープールを使用するその他の例については、「[Amazon Cognito の一般的なシナリオ](#)」を参照してください。

Amazon Cognito では、[責任分担モデル](#)によるクラウド義務のセキュリティは、SOC 1~3、PCI DSS、ISO 27001 に準拠し、HIPAA-BAA に対応しています。Amazon Cognito では、SOC1~3、ISO 27001、HIPAA-BAA に準拠し、PCI DSS に準拠し、PCI DSS に準拠し、Amazon Cognito でクラウド内のセキュリティを設計できます。詳細については、[\[AWS 対象範囲内のサービス\]](#)を参照してください。[リージョンデータに関する考慮事項](#)も参照してください。

Amazon Cognito の機能

ユーザープール

Amazon Cognito ユーザープールは、ユーザーディレクトリです。ユーザープールを使用することで、ユーザーはウェブまたはモバイルアプリに Amazon Cognito 経由でサインインする、またはサードパーティー IdP 経由でフェデレートすることができます。フェデレーションとローカルユーザーには、ユーザープールにユーザープロフィールがあります。

ローカルユーザーとは、ユーザーがユーザープールにサインアップした、または直接作成したユーザーです。これらのユーザープロフィールは、AWS SDK AWS マネジメントコンソール、または AWS Command Line Interface () で管理およびカスタマイズできますAWS CLI。

Amazon Cognito ユーザープールは、サードパーティー IdPs からのトークンとアサーションを受け取り、ユーザー属性を JWT に収集してアプリに発行します。アプリを 1 つの JWT セットで標準化できます。一方、Amazon Cognito が IdP とのインタラクションを処理し、IdPs のクレームを中央トークン形式にマッピングします。

Amazon Cognito ユーザープールは、スタンドアロンIdP である可能性があります。Amazon Cognito は OpenID Connect (OIDC) 標準に基づいて、認証と認可のための JWT を生成しています。ローカルユーザーをログインさせると、ユーザープールがそれらのユーザーに対して権限を持つことになります。ローカルユーザーを認証すると、次の機能にアクセスできます。

- Amazon Cognito ユーザープール API を呼び出してユーザーを認証、認可、管理する独自のウェブフロントエンドを実装します。
- ユーザーに多要素認証 (MFA) を設定します。Amazon Cognito は、タイムベースドワンタイムパスワード (TOTP) と SMS メッセージ MFA をサポートしています。
- 悪意のある管理下にあるユーザーアカウントからのアクセスから保護します。
- 独自のカスタムマルチステップ認証フローを作成します。
- 別のディレクトリでユーザーを検索し、Amazon Cognito に移行します。

Amazon Cognito ユーザープールは、IdP のサービスプロバイダー (SP) とアプリの IdPs という二重の役割を果たすこともできます。Amazon Cognito ユーザープールは、Facebook や Google などのコンシューマー IdPs や、Okta やアクティブディレクトリフェデレーションサービス (ADFS) などのワークフォース IdPs に接続できます。

Amazon Cognito ユーザープールが発行する OAuth 2.0 トークンと OpenID Connect (OIDC) トークンを使用すると、次のことができるようになります。

- ユーザーを認証し、ユーザープロフィールの設定に必要な情報を提供する ID トークンをアプリ内で受け入れる
- ユーザーの API 呼び出しを許可する OIDC スコープを使用して、API でアクセストークンを受け入れます。
- Amazon Cognito ID プールから AWS 認証情報を取得します。

Amazon Cognito ユーザープールの機能

機能	説明
OIDC identity provider	Issue ID tokens to authenticate users
Authorization server	Issue access tokens to authorize user access to APIs
SAML 2.0 service provider	Transform SAML assertions into ID and access tokens
OIDC relying party	Transform OIDC tokens into ID and access tokens

Social provider relying party	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 authentication API requests in supported AWS SDKs ¹
Multi-factor authentication	Use SMS messages, TOTP, 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 ²
Groups	Create logical groupings of users, and a hierarchy of IAM role claims when you pass tokens to identity pools
Customize tokens	Customize your ID and access tokens with new, modified, and suppressed claims
Customize user attributes	Assign values to user attributes and add your own custom attributes

¹ フェデレーションユーザーは、この機能を使用できません。

² フェデレーションユーザーとマネージドログインユーザーは、この機能を使用できません。

ユーザープールの詳細については、「[ユーザープールの開始方法](#)」と[Amazon Cognito ユーザープール API リファレンス](#)を参照してください。

アイデンティティプール

アイデンティティプールは、ユーザーまたはゲストに割り当てて一時的な AWS 認証情報を受け取ることを許可する一意の識別子 (ID) の集まりです。SAML 2.0、OpenID Connect (OIDC)、または OAuth 2.0 ソーシャル ID プロバイダー (IdP) からの信頼できるクレームという形でアイデンティティプールに認証証明書を提示すると、ユーザーをアイデンティティプールの ID に関連付けます。ID プールが ID 用に作成するトークンは、AWS Security Token Service () から一時的なセッション認証情報を取得できますAWS STS。

認証されたアイデンティティを補完するために、IdP 認証なしで AWS アクセスを認可するようにアイデンティティプールを設定することもできます。[デベロッパーが認証した ID](#) を使用して、カスタム認証証明を提供できます。[認証されていない ID](#) を使用して、ゲストユーザーに一時的な AWS 認証情報を付与することもできます。

アイデンティティプールでは、AWS アカウントにおける IAM ポリシーとの統合方法が 2 つあります。これら 2 つの機能は一緒に使用することも、個別に使用することもできます。

ロールベースアクセスコントロール

ユーザーがアイデンティティプールにクレームを渡すと、Amazon Cognito はリクエストする IAM ロールを選択します。ロールの権限をニーズに合わせてカスタマイズするには、各ロールに IAM ポリシーを適用します。たとえば、ユーザーがマーケティング部門に所属していることを証明すると、マーケティング部門のアクセスニーズに合わせたポリシーが設定されたロールの認証情報を受け取ります。Amazon Cognito では、デフォルトロール、ユーザーのクレームをクエリするルールに基づくロール、またはユーザープール内のユーザーのグループメンバーシップに基づくロールをリクエストできます。IAM がユーザーのアイデンティティプールのみを信頼して一時的なセッションを生成するようにロールの信頼ポリシーを設定することもできます。

アクセスコントロールの属性

アイデンティティプールはユーザーのクレームから属性を読み取り、ユーザーの一時セッションのプリンシパルタグにマッピングします。次に、アイデンティティプールからセッションタグを伝達する IAM プリンシパルに基づいてリソースへのアクセスを許可または拒否するように IAM リソーススペースのポリシーを設定できます。たとえば、ユーザーがマーケティング部門にいて実証した場合、はセッション に AWS STS タグを付けますDepartment: marketing。Amazon S3 バケットでは、Department タグの marketing の値が必要な [aws: PrincipalTag](#) 条件に基づく読み取りオペレーションが許可されます。

Amazon Cognito アイデンティティプールの機能

機能	説明
Amazon Cognito user pool relying party	Exchange an ID token from your user pool for web identity credentials from AWS STS
SAML 2.0 service provider	Exchange SAML assertions for web identity credentials from AWS STS
OIDC relying party	Exchange OIDC tokens for web identity credentials from AWS STS
Social provider relying party	Exchange OAuth tokens from Amazon, Facebook, Google, Apple, and Twitter for web identity credentials from AWS STS
Custom relying party	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

アイデンティティプールの詳細については、「[Amazon Cognito アイデンティティプールの開始方法](#)」と「[Amazon Cognito ID プール API リファレンス](#)」を参照してください。

Amazon Cognito ユーザープールとアイデンティティプールの比較

機能	説明	ユーザープール	アイデンティティプール
OIDC identity provider	Issue OIDC ID tokens to authenticate app users	✓	
User directory	Store user profiles for authentication	✓	
Authorize API access	Issue access tokens to authorize user access to APIs (including user profile self-service API operations), databases, and other resources that accept OAuth scopes	✓	
IAM web identity authorization	Generate tokens that you can exchange with AWS STS for temporary AWS credentials		✓
SAML 2.0 service provider & OIDC identity provider	Issue customized OIDC tokens based on claims from a SAML 2.0 identity provider	✓	
OIDC relying party & OIDC identity provider	Issue customized OIDC tokens based	✓	

	on claims from an OIDC identity provider	
OAuth 2.0 relying party & OIDC identity provider	Issue customized OIDC tokens based on scopes from OAuth 2.0 social providers like Apple and Google	✓
SAML 2.0 service provider & credentials broker	Issue temporary AWS credentials based on claims from a SAML 2.0 identity provider	✓
OIDC relying party & credentials broker	Issue temporary AWS credentials based on claims from an OIDC identity provider	✓
Social provider relying party & credentials broker	Issue temporary AWS credentials based on JSON web tokens from developer applications with social providers like Apple and Google	✓
Amazon Cognito user pool relying party & credentials broker	Issue temporary AWS credentials based on JSON web tokens from Amazon Cognito user pools	✓
Custom relying party & credentials broker	Issue temporary AWS credentials to arbitrary identities, authorize d by developer IAM credentials	✓

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, TOTP, 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	✓
User groups	Create logical groupings of users, and a hierarchy of IAM role claims when you pass tokens to identity pools	✓
Customize tokens	Customize your ID and access tokens with new, modified, and suppressed claims and scopes	✓

AWS WAF web ACLs	Monitor and control requests to your authentication front end 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 role trust to limit access to web identity users	✓
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 フェデレーションユーザーは、この機能を使用できません。

Amazon Cognito の使用開始方法

ユーザープールアプリケーションの例については、「[ユーザープールの開始方法](#)」を参照してください。

アイデンティティプールの概要については、「[Amazon Cognito アイデンティティプールの開始方法](#)」を参照してください。

ユーザープールとアイデンティティプールのガイド付きセットアップエクスペリエンスへのリンクについては、「[Amazon Cognito のガイド付きセットアップオプション](#)」を参照してください。

AWS SDK の使用を開始するには、[AWS 「デベロッパーツール」](#)を参照してください。Amazon Cognito 専用のデベロッパーリソースについては、「[Amazon Cognito のリソース](#)」を参照してください。

Amazon Cognito を使用するには、AWS アカウントが必要です。詳細については、「[の開始方法 AWS](#)」を参照してください。

リージョナルな可用性

Amazon Cognito は、世界中の複数の AWS リージョンで利用できます。各リージョンで、Amazon Cognito は複数のアベイラビリティゾーンに分散されています。これらのアベイラビリティゾーンは物理的に相互に分離されていますが、低レイテンシーで高スループットの冗長性に優れたプライベートネットワーク接続で統合されています。これらのアベイラビリティゾーンにより、AWS はレイテンシーを最小限に抑えながら、Amazon Cognito などのサービスを非常に高いレベルの可用性と冗長性で提供できます。

Amazon Cognito が現在 で利用可能かどうかを確認するには AWS リージョン、[AWS 「リージョン別のサービス」](#)を参照してください。

リージョン API サービスエンドポイントの詳細については、「Amazon Web Services 全般のリファレンス」の「[AWS regions and endpoints](#)」を参照してください。

各リージョンで利用できるアベイラビリティゾーンの数については、「[AWS グローバルインフラストラクチャ](#)」を参照してください。

Amazon Cognito の料金

Amazon Cognito の料金については、「[Amazon Cognito の料金](#)」を参照してください。

Amazon Cognito の一般的な用語と概念

Amazon Cognito は、ウェブアプリケーションおよびモバイルアプリケーションの認証情報を提供します。これは、ID 管理とアクセス管理の一般的な用語に基づいて構築されています。ID とアクセスで一般的に使用される用語については、多くのガイドがあります。いくつか例を挙げます。

- IDPro ナレッジボディの[用語](#)
- [AWS ID サービス](#)
- NIST CSRC の[用語集](#)

次のリストは、Amazon Cognito に固有の用語や、Amazon Cognito の特定のコンテキストがある用語を説明しています。

トピック

- [全般](#)
- [ユーザープール](#)
- [アイデンティティプール](#)

全般

このリストの用語は Amazon Cognito に固有のものではなく、Identity and Access Management 実務者の間で広く認識されているものです。以下は、用語の包括的なリストではなく、このガイド中の特定の Amazon Cognito コンテキストについてのガイドです。

アクセストークン

情報システムにアクセスするためのエンティティの[認可](#)に関する情報を含む JSON ウェブトークン (JWT)。

アプリ、アプリケーション

通常、モバイルアプリケーションを指します。このガイドで、アプリケーションとは、多くの場合、Amazon Cognito に接続するウェブアプリケーションまたはモバイルアプリケーションの略語です。

属性ベースのアクセスコントロール (ABAC)

ユーザーの役職や部門などのユーザーのプロパティに基づいてアプリケーションがリソースへのアクセスを決定するモデル。ABAC を適用するための Amazon Cognito ツールには、ユーザープールの ID トークンとアイデンティティプールの[プリンシパルタグ](#)が含まれます。

認証

情報システムへのアクセスを目的として本人確認を行うプロセス。Amazon Cognito は、サードパーティの ID プロバイダーからの認証証明を受け入れるとともに、ソフトウェアアプリケーションの認証プロバイダーにもなります。

認可

リソースにアクセス許可を付与するプロセス。ユーザープールの[アクセストークン](#)には、アプリケーションがユーザーやシステムに対してリソースへのアクセスを許可するための情報が含まれています。

認可サーバー

[JSON ウェブトークン](#) を生成する OAuth または OpenID Connect (OIDC) システム。Amazon Cognito ユーザープールの[マネージド認可サーバー](#)は、ユーザープールの 2 つの認証および認可方法の認可サーバーコンポーネントです。ユーザープールは、[SDK 認証](#)で API チャレンジレスポンスフローもサポートしています。

機密アプリケーション、サーバー側アプリケーション

ユーザーがアプリケーションサーバー上のコードとシークレットへのアクセスを使用してリモートに接続するアプリケーション。これは通常、ウェブアプリケーションです。

ID プロバイダー (IdP)

ユーザー ID を保存して検証するサービス。Amazon Cognito は、[外部プロバイダー](#)に認証をリクエストし、アプリケーションへの IdP として機能できます。

JSON ウェブトークン (JWT)

認証されたユーザーに関するクレームを含む JSON 形式のドキュメント。ID トークンはユーザーを認証し、アクセストークンはユーザーを認可し、更新トークンは認証情報を更新します。Amazon Cognito は[外部プロバイダー](#)からトークンを受け取り、アプリケーションまたは AWS STS にトークンを発行します。

Machine to machine (M2M) 認可

ウェブサーバーアプリケーション層など、非ユーザーインタラクティブなマシンエンティティの API エンドポイントへのリクエストを認可するプロセス。ユーザープールは、[アクセストークン](#)

[OAuth 2.0](#) スコープを使用して、クライアント認証情報の付与における M2M 認可を提供します。

多要素認証 (MFA)

ユーザーがユーザー名とパスワードを入力した後に追加の認証を行う必要があるという要件。Amazon Cognito ユーザープールには、[ローカルユーザー](#)用の MFA 機能があります。

OAuth 2.0 (ソーシャル) プロバイダー

[JWT](#) アクセスと更新トークンを提供するユーザープールまたはアイデンティティプールへの IdP。Amazon Cognito ユーザープールは、ユーザーが認証された後にソーシャルプロバイダーとのインタラクションを自動化します。

OpenID Connect (OIDC) プロバイダー

[OAuth](#) 仕様を拡張して ID トークンを提供するユーザープールまたはアイデンティティプールへの IdP。Amazon Cognito ユーザープールは、ユーザーが認証された後に OIDC プロバイダーとのインタラクションを自動化します。

パスキー、WebAuthn

ユーザーのデバイス上の暗号化キー (パスキー) で認証の証明を行う認証形式。ユーザーは、ハードウェアやソフトウェア認証ツールの生体認証または PIN コードメカニズムを通じて本人確認を行います。パスキーは、フィッシング攻撃に耐性があり、特定のウェブサイトやアプリケーションにバインドされているため、パスワードなしで安全に操作できます。Amazon Cognito ユーザープールは、パスキーを使用したサインインをサポートしています。

パスワードなし

ユーザーがパスワードを入力する必要がない認証形式。パスワードなしのサインイン方法には、E メールアドレスと電話番号、およびパスキーに対して送信されるワンタイムパスワード (OTP) が含まれます。Amazon Cognito ユーザープールは、OTP とパスキーを使用したサインインをサポートしています。

パブリックアプリケーション

コードがローカルに保存され、シークレットへのアクセスを持たない、デバイス上で自己完結するアプリケーション。これは通常、モバイルアプリケーションです。

リソースサーバー

アクセスコントロールを持つ API。また、Amazon Cognito ユーザープールは、リソースサーバーを使用して、API を操作するための設定を定義するコンポーネントを記述します。

ロールベースのアクセスコントロール (RBAC)

ユーザーの職能指定に基づいてアクセスを許可するモデル。Amazon Cognito アイデンティティプールは、各 IAM ロールを区別して RBAC を実装します。

サービスプロバイダー (SP)、依拠しているパーティー (RP)

ユーザーが信頼できることをアサートするために IdP に依存するアプリケーション。Amazon Cognito は、外部 IdP の SP として、また、アプリケーションベースの SP の IdP として機能します。

SAML プロバイダー

ユーザーが Amazon Cognito に渡すデジタル署名されたアサーションドキュメントを生成するユーザープールまたはアイデンティティプールへの IdP。

Universally Unique Identifier (UUID)

オブジェクトに適用される 128 ビットのラベル。Amazon Cognito UUID はユーザープールまたはアイデンティティプールごとに一意ですが、特定の UUID 形式に準拠していません。

ユーザーディレクトリ

特定の情報を他のシステムに提供するユーザーとその属性の集合。Amazon Cognito ユーザープールはユーザーディレクトリであり、外部ユーザーディレクトリのユーザーを統合するためのツールでもあります。

ユーザープール

このガイドで、次のリストの用語は、ユーザープールの特定の機能または設定を参照しています。

アダプティブ認証

潜在的な悪意のあるアクティビティを検出し、[ユーザープロフィール](#)に追加のセキュリティを適用する[高度なセキュリティ](#)の機能。

アプリケーションクライアント

ユーザープールの設定を 1 つのアプリケーションへの IdP として定義するコンポーネント。

コールバック URL、リダイレクト URI、リターン URL

[アプリケーションクライアント](#)の設定と、ユーザープールの[認可サーバー](#)へのリクエストのパラメータ。コールバック URL は、[アプリケーション](#)の認証済みユーザーの最初の送信先です。

選択ベースの認証

ユーザープールでの API 認証形式であり、各ユーザーは複数のサインイン方法から選択できます。ユーザー名とパスワード (MFA を使用または不使用)、パスキーによるサインイン、E メールまたは SMS メッセージのワンタイムパスワードによるパスワードなしのサインインなどから選択できます。アプリケーションは、認証オプションのリストをリクエストするか、希望するオプションを宣言することで、ユーザーが選択するプロセスを方向付けることができます。

[クライアントベースの認証](#)と比較してください。

クライアントベースの認証

ユーザープールの API と AWS SDK で構築されたアプリケーションのバックエンドを使用した認証形式。宣言型認証の場合、アプリケーションはユーザーが実行すべきログインタイプを独自に決定し、そのタイプを事前にリクエストします。

[選択ベースの認証](#)と比較してください。

漏えいした認証情報

攻撃者が知っている可能性のあるユーザーパスワードを検出し、[ユーザープロフィール](#)に追加のセキュリティを適用する [高度なセキュリティ](#) 機能。

確認

新しいユーザーがサインインできるように、前提条件が満たされていると判断するプロセス。確認は通常、E メールアドレスまたは電話番号の [検証](#) を通じて行われます。

カスタム認証

[Lambda トリガー](#)を使用した認証プロセスの拡張で、追加のユーザーチャレンジとレスポンスを定義するもの。

デバイス認証

[MFA](#) を、信頼されたデバイスの ID を使用するサインインに置き換える認証プロセス。

ドメイン、ユーザープールドメイン

AWS で [マネージドログインページ](#) をホストするウェブドメイン。独自に所有するドメインで DNS を設定することも、AWS が所有するドメインで識別用のサブドメインプレフィックスを使用することもできます。

エッセンシャルプラン

ユーザープールの最新の開発が含まれた [機能プラン](#)。エッセンシャルプランには、[プラスプラン](#) の自動学習型セキュリティ機能は含まれていません。

外部プロバイダー、サードパーティープロバイダー

ユーザープールと信頼関係がある IdP。ユーザープールは、外部プロバイダーとアプリケーション間の中間エンティティとして機能し、SAML 2.0、OIDC、およびソーシャルプロバイダーによる認証プロセスを管理します。ユーザープールは、外部プロバイダーの認証結果を 1 つの IdP に統合するため、アプリケーションは単一の、OIDC に依拠しているパーティのライブラリで多数のユーザーを処理できます。

機能プラン

ユーザープールに選択できる機能のグループ。機能プランごとに AWS 請求のコストは異なります。新しいユーザープールには、デフォルトで[エッセンシャルプラン](#)が適用されます。

現在のプラン

- [ライトプラン](#)
- [エッセンシャルプラン](#)
- [プラスプラン](#)

フェデレーションユーザー、外部ユーザー

[外部プロバイダー](#)によって認証されたユーザープール内のユーザー。

ホストされた UI (クラシック)、ホストされた UI ページ

ユーザープールドメインにおける認証フロントエンド、依拠しているパーティ、および ID プロバイダーサービスの初期バージョン。ホストされた UI は、基本的な機能セットと、シンプルな外観と操作感を備えています。ロゴ画像ファイルと事前定義済みの CSS スタイルセットを含むファイルをアップロードして、ホステッド UI ブランディングを適用できます。[マネージドログイン](#)と比較してください。

Lambda トリガー

ユーザープールがユーザー認証プロセスのキーポイントで自動的に呼び出すことができる AWS Lambda 関数。Lambda トリガーを使用して認証結果をカスタマイズできます。

ローカルユーザー

ユーザープール[ユーザーディレクトリ](#)内の[ユーザープロファイル](#)で、[外部プロバイダー](#)による認証によって作成されなかったもの。

リンクされたユーザー

ID が[ローカルユーザー](#)とマージされる[外部プロバイダー](#)のユーザー。

ライトプラン

ユーザープールの最初にリリースされた機能が含まれている[機能プラン](#)。ライトプランには、[エッセンシャルプラン](#)の新機能や[プラスプラン](#)の自動学習型セキュリティ機能は含まれていません。

マネージド認可サーバー、ホストされた UI 認可サーバー、認可サーバー

[マネージドログイン](#)のコンポーネントであり、[ユーザープールドメイン](#)で、IdP とアプリケーションとのやり取り用のサービスをホストします。[ホストされた UI](#) は、マネージドログインとは、提供するユーザーインタラクティブ機能が異なりますが、認可サーバー機能は同じです。

マネージドログイン、マネージドログインページ

[ユーザープールドメイン](#)で、ユーザー認証用のサービスをホストする一連のウェブページ。これらのサービスには、[IdP](#)、サードパーティ IdP に[依拠しているパーティ](#)、およびユーザーインタラクティブ認証 UI のサーバーとして動作する機能が含まれています。ユーザープールのドメインを設定すると、Amazon Cognito のすべてのマネージドログインページがオンラインになります。

アプリケーションは、OIDC ライブラリをインポートすることで、ユーザーのブラウザを起動してマネージドログイン UI に誘導し、サインアップ、サインイン、パスワード管理、その他の認証オペレーションを行います。認証後、OIDC ライブラリは認証リクエストの結果を処理できます。

マネージドログイン認証

ユーザーインタラクティブなブラウザページまたは HTTPS API リクエストを使用して、[ユーザープールドメイン](#)のサービスにサインインします。アプリケーションは OpenID Connect (OIDC) ライブラリを使用してマネージドログイン認証を処理します。このプロセスには、[外部プロバイダー](#)によるサインイン、インタラクティブマネージドログインページによるローカルユーザーのサインイン、[M2M 認可](#)が含まれます。クラシックの[ホストされた UI](#) による認証も、この用語に該当します。

[AWS SDK 認証](#)と比較してください。

プラスプラン

ユーザープールの最新の開発と高度なセキュリティ機能を備えた[機能プラン](#)。

SDK 認証、AWS SDK 認証

AWS SDK を使用してアプリケーションのバックエンドに追加できる認証および認可 API オペレーションのセット。この認証モデルには、独自のカスタム構築したログインメカニズムが必要です。API は、[ローカルユーザー](#)と[リンクされたユーザー](#)にサインインできます。

[マネージドログイン認証](#)と比較してください。

脅威保護、高度なセキュリティ機能

ユーザープールでの脅威保護とは、認証および認可メカニズムに対する脅威を軽減するように設計されたテクノロジーを指します。アダプティブ認証、漏えいした認証情報の検出、IP アドレスのブロックリストは、脅威保護のカテゴリに含まれます。

トークンのカスタマイズ

実行時にユーザーの ID またはアクセストークンを変更するトークン生成前の [Lambda トリガー](#)の結果。

ユーザープール、Amazon Cognito ID プロバイダー、**cognito-idp**、Amazon Cognito ユーザープール

OIDC IdP と連携するアプリケーションの認証および認可サービスを持つ AWS リソース。

検証

ユーザーが E メールアドレスまたは電話番号を所有していることを確認するプロセス。ユーザープールは、新しい E メールアドレスまたは電話番号を入力したユーザーにコードを送信します。Amazon Cognito にコードを送信すると、ユーザーがメッセージの送信先を所有していることを検証して、ユーザープールから追加のメッセージを受信できます。また、「[確認](#)」を参照してください。

ユーザープロフィール、ユーザーアカウント

[ユーザーディレクトリ](#)内のユーザーのエントリ。サードパーティ IdP のユーザーを含むすべてのユーザーは、ユーザープールにプロフィールを持ちます。

アイデンティティプール

このガイドで、次のリストの用語は、アイデンティティプールの特定の機能または設定を参照しています。

アクセスコントロールの属性

アイデンティティプールでの[属性ベースのアクセスコントロール](#)の実装。アイデンティティプールは、ユーザー認証情報にユーザー属性をタグとして適用します。

基本 (クラシック) 認証

[ユーザー認証情報](#)のリクエストをカスタマイズできる認証プロセス。

デベロッパーが認証した ID

[デベロッパー認証情報](#)を使用してアイデンティティプールの[ユーザー認証情報](#)を認可する認証プロセス。

デベロッパー認証情報

アイデンティティプール管理者の IAM API キー。

拡張認証

IAM ロールを選択し、アイデンティティプールで定義したロジックに従ってプリンシパルタグを適用する認証フロー。

ID

アプリケーションユーザーとその[ユーザー認証情報](#)を、アイデンティティプールと信頼関係がある外部[ユーザーディレクトリ](#)内のプロファイルにリンクする [UUID](#)。

アイデンティティプール、Amazon Cognito フェデレーティッド ID、Amazon Cognito ID、**cognito-identity**

[一時的な AWS 認証情報](#)を使用するアプリケーションの認証および認可サービスを持つ AWS リソース。

認証されていない ID

アイデンティティプール IdP でサインインしていないユーザー。認証前に、ユーザーが単一の IAM ロールに対して制限付きユーザー認証情報を生成することを許可できます。

ユーザー認証情報

アイデンティティプール認証後にユーザーが受け取る一時的な AWS API キー。

の開始方法 AWS

Amazon Cognito の使用を開始する前に、必要な AWS リソースをいくつか用意してください。に既にサインインできる場合は AWS アカウント、このセクションをスキップできます。AWS 認証情報

を使用したサインアップとサインインに関する情報をお探しの場合は、引き続きお読みください。十分な AWS Identity and Access Management (IAM) アクセス許可を持つ認証情報を取得したら、[ユーザープール](#)と [ID プール](#)の使用を開始できます。

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. <https://portal.aws.amazon.com/billing/signup> を開きます。
2. オンラインの手順に従います。

サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

にサインアップすると AWS アカウント、AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、[ルートユーザーアクセスが必要なタスク](#)の実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。<https://aws.amazon.com/> の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 のセキュリティを確保し AWS IAM アイデンティティセンター、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、AWS アカウント E メールアドレスを入力して、アカウント所有者[AWS マネジメントコンソール](#)として にサインインします。次のページでパスワードを入力します。

ルートユーザーを使用してサインインする方法については、「AWS サインイン ユーザーガイド」の「[ルートユーザーとしてサインインする](#)」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

手順については、IAM [ユーザーガイドの AWS アカウント](#) 「[ルートユーザー \(コンソール\) の仮想 MFA デバイス](#)」を有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[AWS IAM アイデンティティセンターの有効化](#)」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

を ID ソース IAM アイデンティティセンターディレクトリとして使用する方法のチュートリアルについては、「AWS IAM アイデンティティセンター ユーザーガイド」の「[デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ](#)」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「[ユーザーガイド](#)」の AWS 「[アクセスポータルにサインイン](#)する」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セットを作成する](#)」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[グループを追加する](#)」を参照してください。

ユーザープールの開始方法

アプリケーションで認証とアクセスコントロールを必要とする場合は、シングルサインオン (SSO) 用の OpenID Connect (OIDC) フレームワーク内で作業できます。Amazon Cognito には、AWS SDK を使用してアプリケーションのバックエンドで認証のロジックを処理し、クライアントでブラウザを呼び出してマネージド認可サーバーにアクセスするためのツールがあります。

Amazon Cognito コンソールでは、用意された手順に従って、任意のアプリケーションフレームワークのビューからユーザープールを作成できます。さらに続けて、外部の[ソーシャル](#) ID プロバイダー (IdP) や [SAML 2.0](#) ID プロバイダーとのフェデレーティッドサインインなどの機能を追加できます。Amazon Cognito コンソールのアプリケーションモデルは、プロジェクトへの OIDC ライブラリの追加とブラウザの起動に依存しています。

機能セットを拡張し、Amazon Cognito のコンポーネントをさらに組み込むには、ユーザープールを使用して実行できるすべての事項に関する [Amazon Cognito ユーザープール](#) の章をお読みください。

この章と Amazon Cognito コンソールの例では、アプリケーションリソースと Amazon Cognito ユーザープールとの基本的な統合を示しています。後で、利用可能なオプションで使用するものを増やすようにユーザープールを調整できます。次に、新しい機能の導入や IdP とのやり取りを行うように、アプリケーションを更新できます。

[マネージドログインページ](#) を使用しない場合は、AWS SDK または [AWS Amplify](#) を使用して、カスタムビルドの認証インターフェイスを持つアプリケーションを作成できます。この方法で構築したアプリケーションは、[ユーザープール API](#) とのやり取り用であり、[ローカルユーザー](#) の認証にのみ適しています。この認証モデルの詳細については、「[その他のアプリケーションオプション](#)」を参照してください。

トピック

- [Amazon Cognito コンソールで新しいアプリケーションを作成する](#)
- [その他のアプリケーションオプション](#)
- [ユーザープールに機能とセキュリティオプションを追加する](#)

Amazon Cognito コンソールで新しいアプリケーションを作成する

ユーザープールは、ソフトウェアアプリケーションに認証オプションを追加します。最も簡単に作成を開始するには、Amazon Cognito コンソールにアクセスし、表示される手順に従ってください。作

成手順に従って、ユーザープールリソースの設定だけでなく、アプリケーションの初期設定も行うことができます。

開始する準備ができたなら、[Amazon Cognito コンソール](#)にアクセスし、ボタンを選択して新しいユーザープールを作成します。セットアッププロセスでは、設定とプログラミング言語のオプションについて説明します。

認証の概念に関するその他のリソース

- [Amazon Cognito ユーザープールによる認証](#)
- [API、OIDC、マネージドログインページの認証についての理解](#)
- [Amazon Cognito による認証の仕組み](#)
- [Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#)

アプリケーション用の Amazon Cognito リソースを作成するには

1. [Amazon Cognito コンソール](#) に移動します。Amazon Cognito リソースを作成および管理できるように IAM プリンシパルにアクセス許可を割り当てるには、「[AWS Amazon Cognito の マネージドポリシー](#)」を参照してください。ユーザープールの作成には、AmazonCognitoPowerUser ポリシーで十分です。
2. [ユーザープール] メニューから [ユーザープールを作成] を選択するか、[5 分未満で無料で開始できます] を選択します。
3. [アプリケーションを定義] で、認証および認可サービスを作成する対象のアプリケーションシナリオに最適なアプリケーションタイプを選択します。
4. [アプリケーションに名前を付ける] で、わかりやすい名前を入力するか、デフォルト名を使用します。
5. [オプションを設定] では、ユーザープールの作成後に変更できない設定をサポートする基本的な選択を行う必要があります。
 - a. [サインイン識別子のオプション] で、サインインさせるユーザーを識別する方法を指定します。ユーザー生成のユーザー名、E メールアドレス、または電話番号を選択できます。複数のオプションの組み合わせを許可することもできます。Amazon Cognito は、ここで設定したオプションを、[マネージドログイン](#)のサインインフォームのユーザー名フィールドで受け入れます。

- b. [サインアップのための必須属性] で、ユーザーが新しいアカウントに登録したときに収集するユーザー情報を指定します。マネージドログインページでは、Amazon Cognito はすべての必須属性に対するプロンプトを表示します。

[サインイン識別子のオプション] は、必須属性に影響します。[ユーザー名] には、E メールまたは SMS メッセージでパスワードリセットコードを受信できるように、ユーザーごとに E メールまたは電話が属性として必要です。[メールアドレス] には E メール属性が必要で、[電話番号] には電話番号属性が必要です。

6. [リターン URL を追加] に、ユーザーが認証を完了した後にアプリケーションにリダイレクトするためのパスを入力します。このパスは、OpenID Connect (OIDC) ライブラリを使用してユーザー認証結果を処理するアプリケーション内のルートである必要があります。テストアプリケーション用のリターン URL の例は <https://localhost:3000/callback> です。Amazon Cognito コンソールの NodeJS アプリケーションの例では、このルートは [openid-client](#) を使用してアクセストークンを収集し、ユーザー情報と引き換えます。リソースの作成後に、開発プラットフォームの例を参照できるようになります。
7. [アプリケーションを作成] を選択します。Amazon Cognito は、アプリケーションタイプのデフォルト設定を使用して、ユーザープールとアプリケーションクライアントを作成します。初期リソースの作成後に、[\[外部 ID プロバイダー\]](#) や [\[多要素認証 \(MFA\)\]](#) など、その他のオプションを設定できます。
8. [アプリケーションを設定] ページでは、アプリケーションのコード例をすぐに取得できます。新しいユーザープールを確認するには、下にスクロールして [\[概要に移動\]](#) を選択します。
9. 同じユーザープールに、さらにアプリケーションを追加するには、[アプリケーションクライアント] メニューに移動し、新しいアプリケーションクライアントを追加します。これに伴って、アプリケーションの作成を主としたプロセスが繰り返されますが、ここでは新しいアプリケーションクライアントを既存のユーザープールに追加するだけです。

このプロセスでユーザープールと 1 つ以上のアプリケーションクライアントを作成したら、マネージドログインを使用して認証オペレーションのテストを開始できます。これらのクイックスタートオプションは、一般のセルフサインアップに開放されています。コンソールプロセスを使用してテスト環境を作成したら、最終的な設計を本番環境に移行することをお勧めします。Amazon Cognito の機能に慣れるまでの時間を取ってください。次に、本稼働ワークロードに移行するには、カスタム設定を作成し、AWS CloudFormation や などの自動化ツールを使用してデプロイします AWS Cloud Development Kit (AWS CDK)。

Amazon Cognito は、このプロセスで元に戻すことができないデフォルト設定をいくつか作成します。変更できないユーザープール設定とコンソールで選択できるオプションの詳細については、「[ユーザープールとアプリケーションクライアントの設定更新](#)」を参照してください。

設定	効果	変更方法	詳細情報
クライアントシークレット	認証リクエストにはクライアントシークレットハッシュが必要です。	[従来のウェブアプリケーション] または [Machine to Machine アプリケーション] プロファイルを使用して新しいアプリケーションクライアントを作成します。	アプリケーションクライアントによるアプリケーション固有の設定
優先ユーザー名	ユーザープールは preferred_username 属性をエイリアスとして受け入れません。	AWS SDK を使用してプログラムでユーザープールを作成します。	ログイン属性のカスタマイズ
大文字と小文字の区別	ユーザープールのユーザー名では大文字と小文字は区別されません。例えば、JohnD と johnd は同じユーザーと見なされます。	AWS SDK を使用してプログラムでユーザープールを作成します。	ユーザープールの大文字と小文字の区別

その他のアプリケーションオプション

既存のアプリケーション UI を Amazon Cognito の認証と統合できるかもしれません。独自の認証ページが既にあっても、ディレクトリ設定の機能は Amazon Cognito のユーザープールほどではないかもしれません。このタイプのアプリケーションには、さまざまなプログラミング言語の AWS SDKs で Amazon Cognito 統合を使用して、認証コンポーネントを追加または置き換えることができます。以下にいくつか例を示します。

Amazon Cognito コンソールで、この目的のためのユーザープールを作成すると、インタラクティブなサインインページや OpenID Connect (OIDC) サービスをホストする [ユーザープールドメイン](#) が不要になる場合があります。コンソールでユーザープールを作成するプロセスにより、ドメインは自動的に生成されます。このドメインは、ユーザープールの [ドメイン] タブで削除できます。その他のオプションには、AWS SDKs の API リクエストと CLI の自動セットアップオプションを使用して、アプリケーション用の Amazon Cognito AWS Amplify リソースをプログラムで作成することが含まれます。詳細については、「[Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#)」を参照してください。

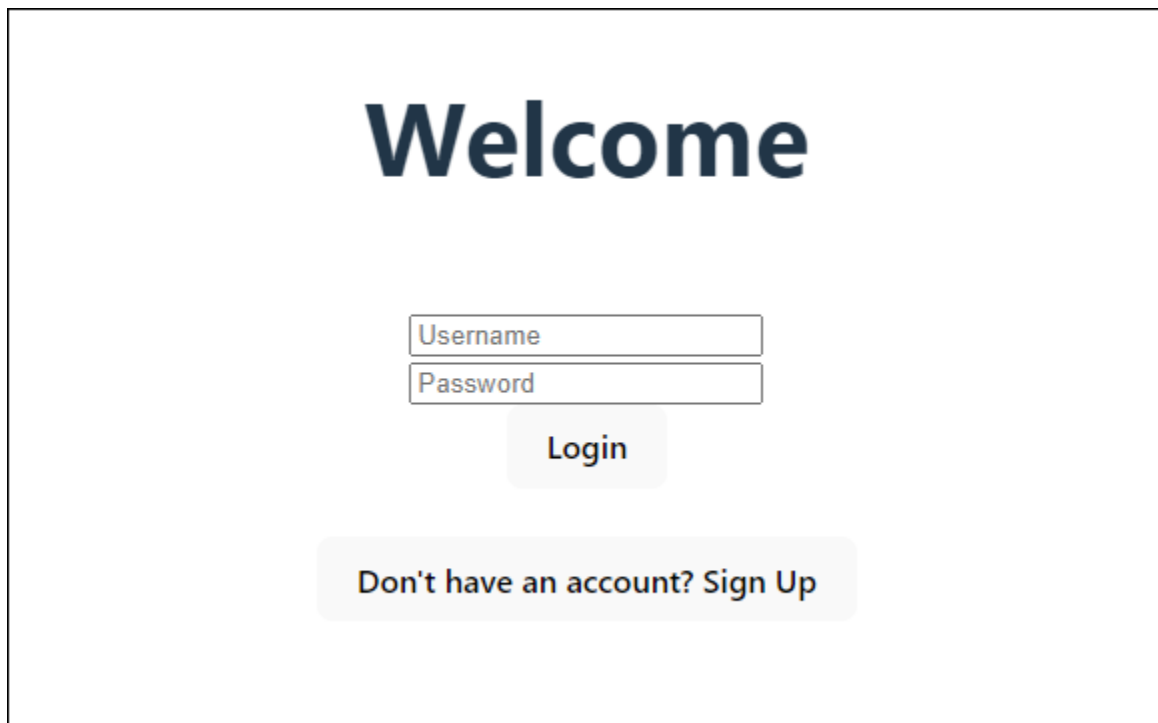
トピック

- [サンプル React 単一ページアプリケーションを設定する](#)
- [Flutter でサンプル Android アプリケーションを設定する](#)

サンプル React 単一ページアプリケーションを設定する

このチュートリアルでは、ユーザーのサインアップ、確認、サインインをテストできる React 単一ページアプリケーションを作成します。React は、ウェブアプリケーションおよびモバイルアプリケーション用の JavaScript ベースのライブラリで、ユーザーインターフェイス (UI) に焦点を当てています。このサンプルアプリケーションは、Amazon Cognito ユーザープールのいくつかの基本的機能を示しています。React を使用したウェブアプリケーション開発の経験がある場合は、[GitHub からサンプルアプリケーションをダウンロード](#)します。

次のスクリーンショットは、作成するアプリケーションの初期認証ページです。



>Welcome

Username

Password

Login

Don't have an account? Sign Up

このアプリケーションをセットアップする場合、ユーザープールは以下の要件を満たしている必要があります。

- ユーザーは E メールアドレスでサインインできます。[Cognito ユーザープールのサインインオプション]: [E メール]。
- ユーザー名の大文字と小文字は区別されません。[ユーザー名の要件]: [ユーザー名の大文字と小文字を区別する] が選択されていません。
- 多要素認証 (MFA) は必要ありません。[MFA の強制]: [オプションの MFA]。
- ユーザープールは、E メールメッセージを使用して、ユーザープロフィールの確認用の属性を検証します。[検証する属性]: [E メールのメッセージを送信、E メールアドレスを検証]。
- E メールは唯一の必須の属性です。[必須の属性]: [E メール]。
- ユーザーはユーザープールにサインアップできます。[自己登録]: [自己登録を有効化] が選択されています。
- 最初のアプリケーションクライアントは、ユーザー名とパスワードによるサインインを許可するパブリッククライアントです。[アプリケーションタイプ]: [パブリッククライアント]、[認証フロー]: ALLOW_USER_PASSWORD_AUTH。

アプリケーションの作成

このアプリケーションを構築するには、デベロッパー環境を設定する必要があります。デベロッパー環境の要件は次のとおりです。

1. Node.js がインストールされ、更新されています。
2. Node Package Manager (npm) がインストールされ、バージョン 10.2.3 以上に更新されています。
3. 環境には、ウェブブラウザの TCP ポート 5173 からアクセスできます。

サンプル React ウェブアプリケーションを作成するには

1. デベロッパー環境にサインインし、アプリケーションの親ディレクトリに移動します。

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

2. 新しい React サービスを作成します。

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

3. GitHub の AWS コード例リポジトリから `cognito-developer-guide-react-example` [プロジェクトフォルダ](#) をクローンします。

```
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/
```

4. プロジェクトで、src ディレクトリに移動します。

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

5. `config.json` を編集して、以下の値を置き換えます:

- a. を AWS リージョン コード `YOUR_AWS_REGION` に置き換えます。例: `us-east-1`。

- b. YOUR_COGNITO_USER_POOL_IDを、テスト用に指定したユーザープールの ID に置き換えます。例: us-east-1_EXAMPLE。ユーザープールは、前のステップで AWS リージョン 入力した 必要がある必要があります。
 - c. YOUR_COGNITO_APP_CLIENT_ID を、テスト用に指定したアプリケーションクライアントの ID に置き換えます。例: 1example23456789。アプリケーションクライアントは、前のステップのユーザープールに存在する必要があります。
6. localhost 以外の IP からサンプルアプリケーションにアクセスする場合は、package.json を編集して行 "dev": "vite", を "dev": "vite --host 0.0.0.0", に変更します。
 7. アプリケーションをインストールします。

```
npm install
```

8. アプリケーションを起動します。

```
npm run dev
```

9. ウェブブラウザで http://localhost:5173 または http://[IP address]:5173 のアプリケーションにアクセスします。
10. 有効な E メールアドレスで新しいユーザーをサインアップします。
11. E メールメッセージから確認コードを取得します。確認コードをアプリケーションに入力します。
12. ユーザー名とパスワードを使用してサインインします。

Amazon Lightsail を使用した React デベロッパー環境の作成

このアプリケーションの使用を開始する簡単な方法は、Amazon Lightsail を使用して仮想クラウドサーバーを作成することです。

Lightsail を使用すると、このサンプルアプリケーションの前提条件が事前設定された小型サーバーインスタンスをすばやく作成できます。ブラウザベースのクライアントを使用してインスタンスに SSH を送信し、パブリックまたはプライベートの IP アドレスでウェブサーバーに接続できます。

このサンプルアプリケーション用の Lightsail インスタンスを作成するには

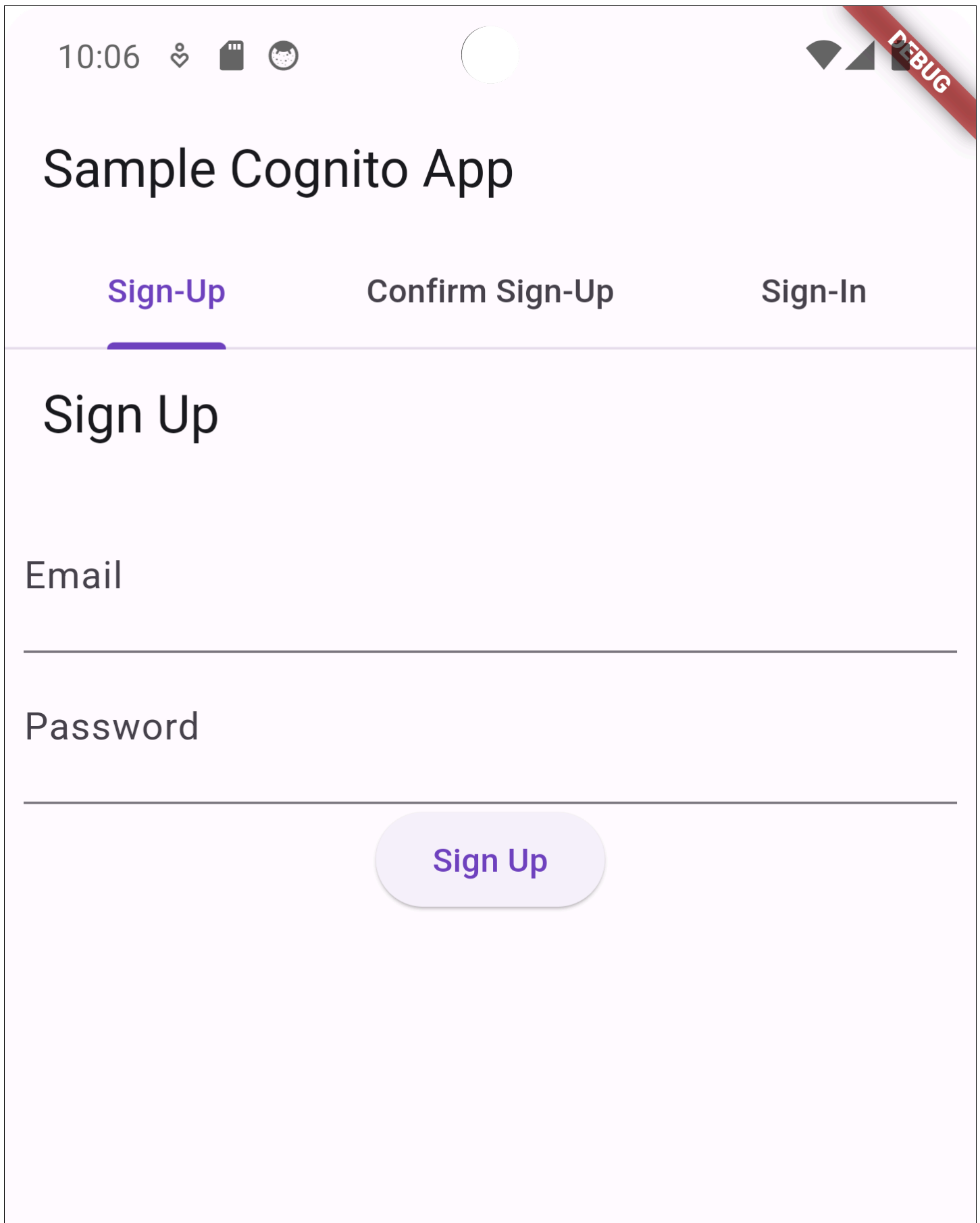
1. [\[Lightsail コンソール\]](#) に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [インスタンスの作成] を選択します。

3. [プラットフォームを選択] で、[Linux/Unix] を選択します。
4. [ブループリントの選択]で、[Node.js] を選択します。
5. [インスタンスを確認] で、開発環境にわかりやすい名前を付けます。
6. [インスタンスの作成] を選択します。
7. Lightsail がインスタンスを作成したら、インスタンスを選択し、[接続] タブから [SSH を使用して接続] を選択します。
8. SSH セッションがブラウザウィンドウで開きます。node -v と npm -v を実行して、インスタンスが Node.js と npm バージョン 10.2.3 以上でプロビジョニングされていることを確認します。
9. [\[React アプリケーションを設定\]](#) に進みます。

Flutter でサンプル Android アプリケーションを設定する

このチュートリアルで、Android Studio でモバイルアプリケーションを作成し、デバイスをエミュレートしてユーザーのサインアップ、確認、サインインをテストできるようになります。このサンプルアプリケーションでは、Flutter で Android 用の基本的な Amazon Cognito ユーザープールモバイルクライアントを作成します。Flutter を使用したモバイルアプリケーション開発の経験がある場合は、[GitHub からサンプルアプリケーションをダウンロード](#)します。

次のスクリーンショットは、仮想 Android デバイスで実行されているアプリケーションを示しています。



このアプリケーションをセットアップするには、ユーザープールが以下の要件を満たしている必要があります。

- ユーザーは E メールアドレスでサインインできます。[Cognito ユーザープールのサインインオプション]: [E メール]。
- ユーザー名の大文字と小文字は区別されません。[ユーザー名の要件]: [ユーザー名の大文字と小文字を区別する] が選択されていません。
- 多要素認証 (MFA) は必要ありません。[MFA の強制]: [オプションの MFA]。
- ユーザープールは、E メールメッセージを使用して、ユーザープロフィールの確認用の属性を検証します。[検証する属性]: [Eメールのメッセージを送信、E メールアドレスを検証]。
- E メールは唯一の必須の属性です。[必須の属性]: [E メール]。
- ユーザーはユーザープールにサインアップできます。[自己登録]: [自己登録を有効化] が選択されています。
- 最初のアプリケーションクライアントは、ユーザー名とパスワードによるサインインを許可するパブリッククライアントです。[アプリケーションタイプ]: [パブリッククライアント]、[認証フロー]: ALLOW_USER_PASSWORD_AUTH。

アプリケーションの作成

サンプル Android アプリケーションを作成するには

1. [\[Android Studio\]](#) と [\[コマンドラインツール\]](#) をインストールします。
2. Android Studio で、[\[Flutter プラグイン\]](#) をインストールします。
3. [このサンプルアプリケーション](#)の `cognito_flutter_mobile_app` ディレクトリの内容から新しい Android Studio プロジェクトを作成します。
 - `assets/config.json` を編集して、`<<YOUR USER POOL ID>>` と `<< YOUR CLIENT ID>>` をユーザープールとアプリケーションクライアントの ID に置き換えます。
4. [Flutter](#) をインストールします。
 - a. PATH 変数に Flutter を追加します。
 - b. 次のコマンドを使用してライセンスを受け入れます。

```
flutter doctor --android-licenses
```
 - c. Flutter 環境を確認し、不足しているコンポーネントをインストールします。

```
flutter doctor
```

- コンポーネントが不足している場合は、`flutter doctor -v` を実行して問題を解決する方法を学びます。
- d. 新しい Flutter プロジェクトのディレクトリに変更し、依存関係をインストールします。
 - `flutter pub add amazon_cognito_identity_dart_2` を実行します。
 - e. `flutter pub add flutter_secure_storage` を実行します。
5. Android 仮想デバイスを作成します。
 1. Android Studio GUI で、[\[デバイスマネージャー\]](#) を使用して新しいデバイスを作成します。
 2. CLI で、`flutter emulators --create --name android-device` を実行します。
 6. 仮想 Android デバイスを起動します。
 1. Android Studio GUI で、仮想デバイスの横にある
 の開始] アイコンを選択します。
 2. CLI で、`flutter emulators --launch android-device` を実行します。
 7. 仮想デバイスでアプリケーションを起動します。
 1. Android Studio GUI で、
 のデプロイ] アイコンを選択します。
 2. CLI で、`flutter run` を実行します。
 8. Android Studio で、実行中の仮想デバイスに移動します。
 9. 有効な E メールアドレスで新しいユーザーをサインアップします。
 10. E メールメッセージから確認コードを取得します。確認コードをアプリケーションに入力します。
 11. ユーザー名とパスワードを使用してサインインします。

ユーザープールに機能とセキュリティオプションを追加する

チュートリアルに従ってサンプルアプリケーションを完成させたら、ユーザープールの実装の範囲を広げることができます。または、テストアプリケーションを作成していない場合は、設定に従って新しいユーザープールを作成します。他のアプリケーションのユーザープール機能をカスタマイズしたり、[外部 ID プロバイダーを追加](#)したりできます。Amazon Cognito ユーザープールを本稼働アプリケーションに配置する移動を計画するときは、[追加のサンプルとチュートリアル](#)を評価できます。

次の優先順位として、ユーザープールでアプリケーションセキュリティオプションを調べて適用する場合は、「[Amazon Cognito ユーザープールに対するセキュリティのベストプラクティス](#)」を参照してください。

Amazon Cognito には機能プランがあり、上位階層にオプトインして機能オプションやセキュリティオプションを追加できます。ライトプランから開始して、エッセンシャルプランで高度な認証や認可のオプションを追加し、プラスプランで自動推論セキュリティガードレールを追加できます。詳細については、「[ユーザープールの機能プラン](#)」を参照してください。

以下は、Amazon Cognito ユーザープールのその他の機能です。

- [マネージドログインページにブランディングを適用する](#)
- [ユーザープールに MFA を追加します](#)
- [脅威保護を備えた高度なセキュリティ](#)
- [Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#)
- [ユーザープール分析に Amazon Pinpoint を使用する](#)

Amazon Cognito の認証と認可のモデルの概要については、「[Amazon Cognito による認証の仕組み](#)」を参照してください。

ユーザープール認証が成功 AWS のサービスした後に他の にアクセスするには、「」を参照してください[サインイン後の ID プール AWS のサービス を使用した へのアクセス](#)。

AWS マネジメントコンソールとユーザープール SDKs を使用するだけでなく、を使用してユーザープールを管理することもできます[AWS Command Line Interface](#)。

トピック

- [ユーザープールにソーシャルサインインを追加する](#)
- [SAML 2.0 ID プロバイダー \(IdP\) を追加する](#)

ユーザープールにソーシャルサインインを追加する

既存のパブリック ID プロバイダーまたはソーシャル ID プロバイダーを通じてアプリケーションにサインインする機能をユーザーに提供することで、認証エクスペリエンスを改善できます。Amazon Cognito ユーザープールは、Facebook、Google、Amazon、Apple などの一般的なソーシャル ID プロバイダー (IdP) と統合され、ユーザーが既に使い慣れている便利なサインインオプションを提供します。

ソーシャルサインインを設定すると、アプリケーション専用のアカウントを作成する代わりにサインインをユーザーが使用できるようになります。これにより、変換率が向上し、サインアッププロセスが一層シームレスになります。ユーザーの視点から見ると、別のユーザー名とパスワードを記憶しておく手間なく、既存のソーシャル認証情報を適用してすばやく認証できます。

ユーザープールでソーシャル IdP を設定するには、いくつかの重要なステップが必要です。クライアント ID とシークレットを取得するには、アプリケーションをソーシャルプロバイダーに登録する必要があります。次に、ソーシャル IdP 設定をユーザープールに追加し、リクエストするスコープと、IdP 属性からマッピングするユーザープール属性を指定できます。実行時に、Amazon Cognito はプロバイダーとのトークン交換を処理し、ユーザー属性をマッピングし、共有ユーザープール形式でアプリケーションにトークンを発行します。

ソーシャル IdP に登録する

Amazon Cognito でソーシャル IdP を作成する前に、アプリケーションをソーシャル IdP に登録して、クライアント ID とクライアントシークレットを取得する必要があります。

アプリケーションを Facebook に登録する

1. [Facebook の開発者アカウント](#)を作成します。
2. Facebook 認証情報を使用して[サインイン](#)します。
3. [My Apps] (マイアプリ) メニューから、[新しいアプリを作成] (新しいアプリを作成) を選択します。

既存の Facebook アプリケーションがない場合は、別のオプションが表示されます。[Create App (アプリの作成)] を選択します。

4. [アプリの作成] ページで、アプリのユースケースを選択し、[次へ] を選択します。
5. Facebook アプリケーションに名前を入力して、[アプリの作成] を選択します。
6. 左のナビゲーションバーで、[設定]、[基本] の順に選択します。

7. [App ID] (アプリ ID) と [App Secret (アプリシークレット)] を書き留めます。これらは次のセクションで使用します。
8. ページの下部で、[+ プラットフォームを追加] を選択します。
9. [プラットフォームの選択] 画面で、プラットフォームを選択し、[次へ] を選択します。
10. [Save changes] (変更の保存) をクリックします。
11. [App Domains] (アプリケーションドメイン) で、ユーザープールのドメインを入力します。

```
https://your_user_pool_domain
```

12. [Save changes] (変更の保存) をクリックします。
13. ナビゲーションバーで [製品] を選択し、[Facebook でログイン] の [設定] を選択します。
14. [Facebook でログイン] の [設定] メニューから [設定] を選択します。

[Valid OAuth Redirect URIs] (有効な OAuth リダイレクト URI) にリダイレクト URL を入力します。リダイレクト URL は、/oauth2/idpresponse エンドポイントを持つユーザープールのドメインで構成されます。

```
https://your_user_pool_domain/oauth2/idpresponse
```

15. [Save changes] (変更の保存) をクリックします。

アプリケーションを Amazon に登録する

1. [Amazon の開発者アカウント](#)を作成します。
2. Amazon 認証情報を使用して[サインイン](#)します。
3. Amazon クライアント ID およびクライアントシークレットを受け取るには、Amazon セキュリティプロファイルを作成する必要があります。

ページの上部にあるナビゲーションバーで [アプリケーションとサービス] を選択し、[Login with Amazon] を選択します。

4. [Create a New Security Profile (新しいセキュリティプロファイルの作成)] を選択します。
5. [Security Profile Name] (セキュリティプロファイル名)、[Security Profile Description] (セキュリティプロファイルの説明)、[Consent Privacy Notice URL] (プライバシー規約 URL の同意) に入力します。
6. [保存] を選択します。

7. [クライアント ID] および [クライアントシークレット] を選択して、クライアント ID およびシークレットを表示します。これらは次のセクションで使用します。
8. 歯車アイコンにマウスカーソルを合わせ、[Web Settings] (ウェブ設定)、[Edit] (編集) の順に選択します。
9. [Allowed Origins] (許可されたオリジン) にユーザープールのドメインを入力します。

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

10. /oauth2/idpresponse エンドポイントを使用するユーザープールドメインを [Allowed Return URLs] (許可されたリターン URL) に入力します。

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

11. [保存] を選択します。

アプリケーションを Google に登録する

Google Cloud プラットフォームの OAuth 2.0 の詳細については、「Google Workspace デベロッパーガイド」ドキュメントの「[認証と認可の詳細](#)」を参照してください。

1. [Google の開発者アカウント](#)を作成します。
2. [Google Cloud Platform コンソール](#)にサインインします。
3. 上部のナビゲーションバーから、[Select a project] (プロジェクトの選択) を選択します。Google プラットフォームにプロジェクトが既にある場合は、このメニューには代わりにデフォルトのプロジェクトが表示されます。
4. [NEW PROJECT] (新しいプロジェクト) を選択します。
5. 製品の名前を入力し、[CREATE] (作成) を選択します。
6. 左のナビゲーションバーで、[API とサービス] を選択し、[OAuth 同意画面] を選択します。
7. アプリケーション情報である [アプリケーションドメイン]、[認可済みドメイン]、[デベロッパーの連絡先情報] を入力します。[認可済みドメイン] には、amazoncognito.com とカスタムドメインのルートを含める必要があります。例: example.com。[SAVE AND CONTINUE] (保存して続行) を選択します。
8. 1. [スコープ] で、[スコープの追加または削除] を選択し、少なくとも、次の OAuth スコープを選択します。

1. .../auth/userinfo.email

2. .../auth/userinfo.profile
3. openid
9. [Test users] (テストユーザー) で、[Add users] (ユーザーの追加) を選択します。E メールアドレスおよびその他の認可済みテストユーザーを入力して、[保存して続行] を選択します。
10. 左のナビゲーションバーを再度展開し、[API とサービス] を選択して、[認証情報] を選択します。
11. [認証情報の作成]、[OAuth クライアント ID] の順に選択します。
12. [Application type] (アプリケーションタイプ) を選択し、クライアントに [Name] (名前) を入力します。
13. [Authorized JavaScript origins] (承認済みの JavaScript 生成元) で、[ADD URI] (URI の追加) を選択します。ユーザープールのドメインを入力します。

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

14. [Authorized redirect URIs] (承認済みのリダイレクト URI) で、[ADD URI] (URI の追加) を選択します。ユーザープールのドメインの /oauth2/idpresponse エンドポイントへのパスを入力します。

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

15. [作成] を選択します。
16. Google が [クライアント ID] と [クライアントシークレット] の下に表示する値を安全に保存します。Google IdP を追加するときに、Amazon Cognito にこれらの値を指定します。

アプリケーションを Apple に登録する

「Apple でサインイン」を設定する方法についての詳細は、Apple 開発者ドキュメントの「[Configuring Your Environment for Sign in with Apple](#)」(Apple でサインインするための環境を設定する) を参照してください。

1. [Apple の開発者アカウント](#)を作成します。
2. Apple 認証情報を使用して[サインイン](#)します。
3. 左のナビゲーションバーで、[Certificates, Identifiers & Profiles] (証明書、ID & プロファイル) を選択します。
4. 左のナビゲーションペインで、[Identifiers] (識別子) を選択します。

5. [Identifiers] (識別子) ページで、[+] アイコンを選択します。
6. [Register a New Identifier] (新しい識別子の登録) ページで、[App IDs] (アプリ ID)、[Continue] (続行) の順に選択します。
7. [種類の選択] ページで、[アプリケーション] を選択し、[続行] を選択します。
8. [Register an App ID] ページで、次の操作を行います。
 1. [Description] (説明) で、説明を入力します。
 2. [App ID Prefix] (アプリ ID プレフィックス) に、[Bundle ID] (バンドル ID) を入力します。[App ID Prefix] (アプリケーション ID プレフィックス) にある値を書き留めておきます。この値は、[ソーシャル IdP を使用してユーザープールを設定する](#) で Apple を ID プロバイダーとして選択した後で使用します。
 3. [Capabilities] (機能) で、[Sign In with Apple] (Apple でサインイン) を選択してから [Edit] (編集) を選択します。
 4. [Sign in with Apple: App ID Configuration] (Apple でサインイン: アプリ ID の設定) ページで、アプリをプライマリとして設定するか、他のアプリ ID とグループ化するかを選択し、次に [Save] (保存) を選択します。
 5. [続行] をクリックしてください。
9. [Confirm your App ID] ページで、[登録] を選択します。
10. [Identifiers] (識別子) ページで、[+] アイコンを選択します。
11. [Register a New Identifier] (新しい識別子の登録) ページで、[Services IDs] (サービス ID)、[Continue] (続行) の順に選択します。
12. [Register a Services ID] ページで、次の操作を行います。
 1. [Description] (説明) で、説明を入力します。
 2. [Identifier] (識別子) で、識別子を入力します。このサービス ID をメモしておきます。この値が、[ソーシャル IdP を使用してユーザープールを設定する](#) で Apple を ID プロバイダーとして選択した後で必要になるからです。
 3. [Continue (続行)] を選択し、[Register (登録)] を選択します。
13. [識別子] ページから、先ほど作成した [サービス ID] を選択します。
 1. [Sign In with Apple] (Apple でサインイン) を選択後、[Configure] (設定) を選択します。
 2. [Web Authentication Configuration] (ウェブ認証の設定) ページで、[Primary App ID] (プライマリアプリ ID) として前に作成したアプリ ID を選択します。
 3. [Website URLs] (ウェブサイトの URL) の横の [+] アイコンを選択します。

4. [Domains and subdomains] (ドメインとサブドメイン) で、https:// プレフィックスなしでユーザープールのドメインを入力します。

`<your-user-pool-domain>`

5. [Return URLs] (URL を返す) で、ユーザープールのドメインの /oauth2/idpresponse エンドポイントへのパスを入力します。

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

6. [次へ] を選択してから、[完了] を選択します。ドメインを検証する必要はありません。
7. [Continue] (続行) を選択し、次に [Save] (保存) を選択します。
14. 左のナビゲーションペインで [Keys] (キー) を選択します。
15. [Keys (キー)] ページで、[+] アイコンを選択します。
16. [Register a New Key] ページで、次の操作を行います。
 1. [Key Name] (キー名) に、キー名を入力します。
 2. [Sign In with Apple] (Apple でサインイン) を選択後、[Configure] (設定) を選択します。
 3. [キーの設定] ページで、[プライマリアプリケーション ID] として前に作成したアプリケーション ID を選択します。[保存] を選択します。
 4. [Continue] (続行) を選択し、[Register] (登録) を選択します。
17. [キーをダウンロード] ページで、[ダウンロード] をクリックしてプライベートキーをダウンロードしてから、表示される [キー ID] を書き留め、[完了] を選択します。このプライベートキーと、このページに表示されている [Key ID] (キー ID) の値は、[ソーシャル IdP を使用してユーザープールを設定する](#) で Apple を ID プロバイダーとして選択した後で必要になります。

ユーザープールにソーシャル IdP を追加する

このセクションでは、前のセクションで取得したクライアント ID およびクライアントシークレットを使用してユーザープールにソーシャル IdP を設定します。

を使用してユーザープールソーシャル ID プロバイダーを設定するには AWS マネジメントコンソール

1. [Amazon Cognito コンソール](#)に移動します。AWS 認証情報の入力を求められる場合があります。

2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択します。[Federated sign-in] (フェデレーションサインイン) を検索し、[Add an identity provider] (ID プロバイダーの追加) を選択します。
5. ソーシャル ID プロバイダー ([Facebook]、[Google]、[Login with Amazon]、[Sign in with Apple]) を選択します。
6. ソーシャル ID プロバイダーの選択に基づいて、次のステップから選択します。
 - Google と Login with Amazon — 前のセクションで生成された [アプリケーションクライアント ID] と [アプリケーションクライアントのシークレット] を入力します。
 - Facebook — 前のセクションで生成された [アプリケーションクライアント ID] と [アプリケーションクライアントのシークレット] を入力し、API バージョン (例えば、バージョン 2.12) を選択します。Facebook API の各バージョンにはライフサイクルとサポート終了日があるため、可能な限り最新のバージョンを選択することをお勧めします。Facebook のスコープと属性は API バージョンによって異なる場合があります。Facebook でソーシャル ID ログインをテストして、フェデレーションが意図したとおりに機能することを確認することをお勧めします。
 - Sign in with Apple — 以前のセクションで生成された [サービス ID]、[チーム ID]、[キー ID] および [プライベートキー] を入力します。
7. 使用する [認可済みスコープ] の名前を入力します。スコープは、アプリでアクセスするユーザー属性 (name や email など) を定義します。Facebook の場合は、コンマで区切る必要があります。Google および Login with Amazon の場合は、スペースで区切って指定します。Sign in with Apple の場合は、アクセスするスコープのチェックボックスをオンにします。

ソーシャル ID プロバイダー	スコープ例
Facebook	public_profile, email
Google	profile email openid
Login with Amazon	profile postal_code
Apple でのサインイン	email name

アプリケーションユーザーは、これらの属性をアプリケーションに提供することに同意するよう求められます。ソーシャルプロバイダーのスコープの詳細については、Google、Facebook、Login with Amazon、および Sign in with Apple のドキュメントを参照してください。

以下は、Sign in with Apple の場合にスコープが返らない可能性があるユーザーシナリオです。

- エンドユーザーが Apple のサインインページを離れるとエラーが発生する (Amazon Cognito の内部障害またはデベロッパーが記述したプログラムが原因である可能性があります)。
 - サービス ID 識別子がユーザープールや他の認証サービス全体で使用されている。
 - ユーザーがサインインした後に、デベロッパーがスコープを追加している。ユーザーは認証時およびトークンの更新時にのみ、新しい情報を取得する。
 - デベロッパーによって削除されたユーザーが Apple ID プロファイルからアプリケーションを削除せずに再度サインインしている。
8. ID プロバイダーからユーザープールに属性をマッピングします。詳細については、「[マッピングについて知っておくべきこと](#)」を参照してください。
 9. [作成] を選択します。
 10. [アプリケーションクライアント] メニューで、リストのアプリケーションクライアントから 1 つを選択し、[ホストされた UI 設定の編集] を選択します。[Identity providers] (ID プロバイダー) で、新しいソーシャル ID プロバイダーをアプリケーションクライアントに追加します。
 11. [Save changes] (変更の保存) をクリックします。

ソーシャル IdP の設定をテストする

前の 2 つのセクションの要素を使用してログイン URL を作成できます。この URL を使用してソーシャル IdP の設定をテストします。

```
https://mydomain.auth.us-east-1.amazoncognito.com/login?  
response_type=code&client_id=1example23456789&redirect_uri=https://www.example.com
```

ドメインは、コンソールにあるユーザープールの [ドメイン名] ページで確認できます。クライアント ID は [アプリクライアントの設定] ページにあります。redirect_uri パラメータのコールバック URL を使用します。これは、ユーザーが認証に成功した後でリダイレクトされるページの URL です。

 Note

Amazon Cognito は 5 分以内に完了しない認証リクエストをキャンセルし、ユーザーをマネージドログインにリダイレクトします。ページには、Something went wrong というエラーメッセージが表示されます。

SAML 2.0 ID プロバイダー (IdP) を追加する

アプリケーションユーザーは SAML 2.0 ID プロバイダー (IdP) を使用してサインインできます。顧客が組織の内部顧客またはリンクされたビジネスである場合は、ソーシャル IdP ではなく SAML 2.0 IdP を選択できます。ソーシャル IdP がすべてのユーザーにアカウントへの登録を許可する場合、SAML IdP は組織が管理するユーザーディレクトリとペアになる可能性が高くなります。ユーザーが直接サインインしても、サードパーティーを介してサインインしても、すべてのユーザーにはユーザープールにプロファイルがあります。SAML ID プロバイダーを通じてサインインを追加しない場合は、この手順を省略してください。

詳細については、「[ユーザープールによる SAML ID プロバイダーの使用](#)」を参照してください。

SAML ID プロバイダーを更新し、ユーザープールを設定する必要があります。ユーザープールを SAML 2.0 ID プロバイダーの依拠しているパーティーまたはアプリケーションとして追加する方法については、SAML ID プロバイダーのドキュメントを参照してください。

また、SAML ID プロバイダーにアサーションコンシューマーサービス (ACS) エンドポイントを提供する必要があります。SAML ID プロバイダーの SAML 2.0 POST バインド用に、ユーザープールドメインで次のエンドポイントを設定します。ユーザープールドメインの詳細については、「[ユーザープールのドメインを設定する](#)」を参照してください。

`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`

ユーザープールのドメインプレフィックスとリージョン値は、[Amazon Cognito コンソール](#)の [ドメイン] メニューにあります。

SAML ID プロバイダーによっては、オーディエンス URI や SP エンティティ ID と呼ばれる、サービスプロバイダー (SP) urn も指定する必要があります。


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

ユーザープール ID は、[Amazon Cognito コンソール](#)のユーザープールに関する [概要] で確認できます。

また、SAML ID プロバイダーを設定して、ユーザープールに必要なすべての属性の属性値を提供する必要があります。通常、email はユーザープールの必須属性です。その場合、SAML ID プロバイダーは、SAML アサーションの email 値 (クレーム) を指定する必要があります。

Amazon Cognito ユーザープールはバインディング後のエンドポイントで SAML 2.0 フェデレーションをサポートします。これにより、ユーザープールがユーザーエージェント経由で ID プロバイダーから直接 SAML レスポンスを受信するため、アプリケーションで SAML アサーションレスポンスを受信して解析する必要がなくなります。

ユーザープールに SAML 2.0 ID プロバイダーを設定する

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択します。[Federated sign-in] (フェデレーションサインイン) を検索し、[Add an identity provider] (ID プロバイダーの追加) を選択します。
5. [SAML] ソーシャル ID プロバイダーを選択します。
6. カンマで区切られた [Identifiers] (識別子) を入力します。識別子は Amazon Cognito に、ユーザーがサインインしたときに入力したメールアドレスを確認する必要があることを伝えます。次に、ドメインに対応するプロバイダーに誘導します。
7. ユーザーがログアウトしたときに、Amazon Cognito が署名されたサインアウト要求をプロバイダーに送信するためには、[Add sign-out flow] (サインアウトフローの追加) を選択します。SAML 2.0 ID プロバイダーを設定して、マネージドログインの設定時に作成した `https://<your Amazon Cognito domain>/saml2/logout` エンドポイントに、サインアウト応答を送信する必要があります。saml2/logout エンドポイントでは、POST バインディングを使用します。

Note

このオプションが選択されていて、SAML ID プロバイダーが署名付きログアウトリクエストを期待する場合は、SAML IdP で Amazon Cognito が提供する署名証明書を設定する必要もあります。

SAML IdP は、署名されたログアウトリクエストを処理し、Amazon Cognito セッションからユーザーをログアウトさせます。

8. [Metadata document source] (メタデータドキュメントソース) を選択します。ID プロバイダーがパブリック URL で SAML メタデータを提供する場合は、[Metadata document URL] (メタデータドキュメント URL) を選択してそのパブリック URL を入力できます。それ以外の場合は、[Upload metadata document] (メタデータドキュメントをアップロード) を選択し、プロバイダーから以前ダウンロードしたメタデータファイルを選択します。

Note

プロバイダーにパブリックエンドポイントがある場合は、ファイルをアップロードするのではなく、メタデータドキュメントの URL を入力することをお勧めします。これにより、Amazon Cognito はメタデータを自動的に更新できます。通常、メタデータの更新は 6 時間ごとまたはメタデータの有効期限が切れる前のいずれか早いタイミングで発生します。

9. [Map attributes between your SAML provider and your appS] (AML プロバイダーとアプリケーション間で属性をマッピングする) をクリックして、SAML プロバイダー属性をユーザープールのユーザープロファイルにマッピングします。ユーザープールの必須属性を属性マップに含めます。

たとえば、[User pool attribute] (ユーザープール属性) email を選択して、ID プロバイダーからの SAML アサーションに表示される SAML 属性名を入力します。ID プロバイダーは、参考として SAML アサーションのサンプルを提供する場合があります。ID プロバイダーの中には、email などのような単純な名前を使用している ID プロバイダーもあります。次の例のように、URL 形式の属性名を使用するものもあります。

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

10. [Create] (作成) を選択します。

Amazon Cognito アイデンティティプールの開始方法

Amazon Cognito ID プールを使用すると、一意のアイデンティティを作成し、ユーザーに許可を割り当てることができます。アイデンティティプールは、次のタイプの認証サービスから ID を取り込むことができます。

- Amazon Cognito ユーザープール内のユーザー
- Facebook、Google、Apple、または SAML ID プロバイダーなどの外部 ID プロバイダーを使って認証するユーザー。
- 独自の既存の認証プロセスによって認証されたユーザー

ユーザーがプロバイダーで認証し、ID プールに認可を提示すると、一時的な AWS 認証情報を取得します。ユーザーの認証情報には、他の AWS のサービスにアクセスするために定義したアクセス許可が含まれます。

トピック

- [Amazon Cognito で ID プールを作成する](#)
- [SDK の設定](#)
- [ID プロバイダーを統合する](#)
- [認証情報を取得する](#)
- [アイデンティティプールのサンプルアプリケーション](#)

Amazon Cognito で ID プールを作成する

ID プールは、Amazon Cognito コンソール経由で、または、AWS Command Line Interface (CLI) が Amazon Cognito API を使用して作成することができます。次の手順は、コンソールで新しいアイデンティティプールを作成するための一般的なガイドです。[コンソールに直接スキップ](#)して、ガイド付きエクスペリエンスとインラインヘルプコンテンツに従うこともできます。

コンソールで新しい ID プールを作成する

1. [Amazon Cognito コンソール](#)にサインインし、アイデンティティプールを選択します。Amazon Cognito リソースを作成および管理できるように IAM プリンシパルにアクセス許可を割り当てるには、「[AWS Amazon Cognito の マネージドポリシー](#)」を参照してください。アイデンティティプールの作成には AmazonCognitoPowerUser ポリシーで十分です。

2. [ID プールを作成] を選択します。
3. [ID プールの信頼を設定] で、アイデンティティプールを認証アクセス、ゲストアクセス、またはその両方に設定することを選択します。
 - [認証アクセス] を選択した場合、アイデンティティプールの認証済み ID のソースとして設定する ID タイプを 1 つ以上選択します。[カスタムデベロッパープロバイダー] を設定した場合、アイデンティティプールの作成後にそのプロバイダーを変更したり削除したりすることはできません。
4. [許可を設定] で、アイデンティティプール内の認証済みユーザーまたはゲストユーザーのデフォルトの IAM ロールを選択します。
 - a. Amazon Cognito に、基本的な権限とアイデンティティプールとの信頼関係を持つ新しいロールを作成する場合は、[新しい IAM ロールを作成] を選択します。新しいロールを識別するための IAM ロール名を入力します (たとえば `myidentitypool_authenticatedrole`)。[ポリシードキュメントを表示] を選択して、Amazon Cognito が新しい IAM ロールに割り当てるアクセス権限を確認します。
 - b. 使用する にロールがすでにある場合は、既存の IAM ロール AWS アカウント を使用することを選択できます。cognito-identity.amazonaws.com を含めるように IAM ロールの信頼ポリシーを設定する必要があります。リクエストが特定のアイデンティティプール内の認証されたユーザーから送信されたという証拠を提示した場合にのみ、Amazon Cognito がロールを引き継ぐことを許可するようにロールの信頼ポリシーを設定します。詳細については、「[ロールの信頼とアクセス権限](#)」を参照してください。
5. [ID プロバイダーを接続] で、[ID プールの信頼の設定] で選択した ID プロバイダー (IdPs) の詳細を入力します。OAuth アプリケーションクライアント情報の提供、Amazon Cognito ユーザープールの選択、IAM IdP の選択、またはデベロッパープロバイダーのカスタム識別子の入力を求められる場合があります。
 - a. 各 IdP のロール設定を選択します。その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。Amazon Cognito ユーザープール IdP では、トークンに `preferred_role` を含むロールを選択することもできます。cognito:preferred_role クレームの詳細については、「[グループへの優先順位の値の割り当て](#)」を参照してください。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。

- ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
- b. アクセスコントロールの属性は、IdP ごとに個別に設定できます。アクセスコントロールの属性は、Amazon Cognito がユーザーの一時セッションに適用する [プリンシパルタグ](#) にユーザーのクレームをマッピングします。セッションに適用するタグに基づいてユーザーアクセスをフィルタリングするIAMポリシーを作成できます。
 - i. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - ii. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - iii. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
- 6. [プロパティの設定] の [ID プール名] に [名前] を入力します。
- 7. [基本 (クラシック) 認証] で、ベーシックフローを有効にするかどうかを選択します。基本フローがアクティブになっていると、IdPs に対して行ったロール選択をバイパスして [AssumeRoleWithWebIdentity](#) を直接呼び出すことができます。詳細については、「[ID プールの認証フロー](#)」を参照してください。
- 8. アイデンティティプールに [タグ](#) を適用する場合は、[タグ] で [タグの追加] を選択します。
- 9. [確認および作成] で、新しいアイデンティティプールに対して行った選択を確認します。[編集] を選択してウィザードに戻り、設定を変更します。終了したら、[ID プールの作成] を選択します。

SDKE の設定

Amazon Cognito ID プールを使用するには、AWS Amplify、AWS SDK for Java、または を設定します SDK for .NET。詳細については、以下のトピックを参照してください。

- AWS SDK for JavaScript デベロッパーガイドの [SDK for JavaScript のセットアップ](#)
- Amplify Dev Center の「[Amplify のドキュメント](#)」
- [SDK for .NET デベロッパーガイド](#) の「Amazon Cognito 認証情報プロバイダー」

ID プロバイダーを統合する

Amazon Cognito アイデンティティプール (フェデレーテッド ID) は、Amazon Cognito ユーザープール、フェデレーテッド ID プロバイダー (Amazon、Facebook、Google、および SAML ID プロバイダーなど) に加えて、認証されていない ID 経由でのユーザー認証をサポートしています。また、この機能は独自のバックエンド認証プロセスを通じてユーザーを認証し登録することができる、[デベロッパーが認証した ID](#) をサポートしています。

独自のユーザーディレクトリを作成するための Amazon Cognito ユーザープールの使用に関する詳細については、「[Amazon Cognito user pools](#)」および「[サインイン後の ID プール AWS のサービスを使用した へのアクセス](#)」を参照してください。

外部 ID プロバイダーについては、[アイデンティティプールのサードパーティー ID プロバイダー](#) を参照してください。

独自のバックエンド認証プロセスの統合に関する詳細については、「[デベロッパーが認証した ID](#)」を参照してください。

認証情報を取得する

Amazon Cognito ID プールは、ゲストであるユーザー (認証されていないユーザー) と、トークンを認証して受け取ったユーザーに一時的な AWS 認証情報を提供します。これらの AWS 認証情報を使用すると、アプリケーションは Amazon API Gateway AWS AWS を介して内外のバックエンドに安全にアクセスできます。「[認証情報の取得](#)」を参照してください。

アイデンティティプールのサンプルアプリケーション

Amazon Cognito ID プールの最も一般的なユースケースは、複数のサインインシステムからユーザーをフェデレーションし、アクセスが制限された一時的な AWS 認証情報をクライアントに直接配信することです。これにより、AWS リソースにアクセスするためのアクセス許可のための認証情報ブローカーを構築する必要がなくなります。例えば、モバイルアプリで Amazon S3 からソーシャルメディアアカウントを使用してサインインし、アプリアセットにアクセスすることをユーザーに許可する必要が生じることがあります。アイデンティティプールは、ユーザープールでサインインするユーザーに認証情報も配信します。

このチュートリアルでは、アイデンティティプールでサポートされている ID プロバイダー (IdP) を使用して、拡張および基本[認証フロー](#)で一時的な認証済み認証情報とゲスト認証情報を取得できる

ウェブアプリケーションを作成します。ウェブ開発の経験がある場合は、GitHub からサンプルアプリケーションをダウンロードしてください。

[GitHub からサンプルアプリケーションをダウンロードする](#)

このサンプルアプリケーションでは、Amazon Cognito のアイデンティティプールの以下の機能について、わかりやすく説明します。

アイデンティティプールの認証フロー

- API リクエストの詳細な内訳を含む拡張認証フロー
- API リクエストの詳細な内訳を含む基本認証フロー

ゲスト (認証されていない) アクセスの実装

- サインインを必要とせずに制限付き AWS のサービス アクセスを提供する

サポートされている ID プロバイダーとの統合

- コンシューマーアクセス向けのソーシャル IdP (Facebook、Amazon、Twitter、Apple、Google)
- 企業ユーザー向けのエンタープライズ IdP (OpenID Connect または SAML 経由)
- Amazon Cognito ユーザープール

AWS 認証情報管理

- ID プロバイダートークンを一時的な AWS 認証情報と交換する
- 一時的な認証情報を使用して AWS サービスに安全にアクセスする

開発用ウェブサーバーでアプリケーションを設定し、ブラウザからアクセスすると、以下のオプションが表示されます。

Explore

Amazon Cognito identity pools authentication flows

Explore authentication flows in identity pools and how to integrate them with supported identity providers (IdP). This interactive demo application is intended for education purposes only and not for production usage.

01.

Authentication flow overview

Amazon Cognito identity pools support two authentication flows: a basic flow that maps guest or federated users to temporary AWS credentials, and an enhanced flow that first issues an OpenID token so you can scope roles or add session tags before requesting credentials.

02.

Enhanced and basic flow comparison

Compare basic and enhanced flows in identity pools side by side and choose the right approach for your application.

03.

Interactive demo

Test different authentication flows and API calls using supported identity providers to obtain temporary AWS credentials for accessing AWS services.



トピック

- [前提条件](#)
- [認証プロバイダーの設定](#)
- [デモアプリケーションをデプロイする](#)
- [アイデンティティプールの認証方法を調べる](#)
- [次の手順](#)

前提条件

始める前に、以下のリソースを設定する必要があります。

- Amazon Cognito にアクセスできる AWS アカウント。AWS アカウントをお持ちでない場合は、「」の手順に従います [の開始方法 AWS](#)。
- Python 3.8 以降が開発コンピュータにインストールされていること。
- GitHub へのアクセス権。
- AWS Amazon Cognito APIs に認証されたリクエストを行うアクセス許可で設定された 認証情報。これらの認証情報は、[デベロッパーの認証](#)に必要です。

特定の SDK での AWS 認証情報と ID プールフェデレーションの実装の詳細については、「」を参照してください[the section called “認証情報の取得”](#)。

認証プロバイダーの設定

このアプリケーションで最良の結果を得るには、1 つ以上のサードパーティ ID プロバイダー (IdP) または Amazon Cognito ユーザープールをセットアップし、Amazon Cognito アイデンティティプールと統合します。前提条件を満たしたら、このデモアプリケーションを実行する前に、設定する ID プロバイダーを選択します。[Amazon Cognito コンソール](#)では、アイデンティティプールと ID プロバイダーを設定する手順を示しています。

Amazon Cognito ユーザープール

- [the section called “認証”](#)
- [the section called “アプリクライアント”](#)

ソーシャル ID プロバイダー

- Google: [the section called “Google”](#)
- Facebook: [the section called “Facebook”](#)
- Amazon: [the section called “Login with Amazon”](#)

OpenID Connect (OIDC) プロバイダー

- [the section called “Open ID Connect プロバイダー”](#)

SAML プロバイダー

- [the section called “SAML ID プロバイダー”](#)

Note

このデモアプリケーションでは、サポートされているすべての ID プロバイダーをセットアップする必要はありません。ユースケースに合ったものから開始できます。各リンクには、詳細な設定手順が記載されています。

デモアプリケーションをデプロイする

リポジトリをクローンします

1. ターミナルウィンドウを開きます。

2. aws-doc-sdk-examples リポジトリのクローンを作成するか、[このフォルダをリポジトリから](#)取得します。

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

3. プロジェクトディレクトリに移動します。

```
cd python/example_code/cognito/scenarios/identity_pools_example_demo/web
```

アイデンティティプールを作成する

アプリケーションの Amazon Cognito アイデンティティプールを作成するには、「[the section called “アイデンティティプールの設定”](#)」の手順に従います。

デモアプリケーションのアイデンティティプールを設定するには

1. [Amazon Cognito コンソール](#)を開きます。
2. ナビゲーションペインで、[アイデンティティプール] を選択します。既存のアイデンティティプールを選択するか、新しいアイデンティティプールを作成します。
3. [ユーザーアクセス] で、[認証されたアクセス] と [ゲストアクセス] を有効にします。新規または既存の [IAM ロール](#) を設定し、各タイプのユーザーに付与する [アクセス許可をロールに割り当て](#)ます。
4. [ユーザーアクセス] で、設定する ID プロバイダーをセットアップします。
5. [ID プールのプロパティ] で、[基本 (クラシック) 認証] を有効にします。
6. ブラウザでアイデンティティプールのコンソールを開いたままにします。アイデンティティプール ID とその他の設定情報は、アプリケーションのセットアップで使います。

アプリケーションを設定して実行する

次の手順では、デモアプリケーションの初期セットアップについて説明します。

デモアプリケーションを設定するには

1. aws-doc-sdk-examples クローンの python/example_code/cognito/scenarios/identity_pools_example_demo/web でコマンドラインを開きます。
2. [サンプル環境ファイル](#)をコピーして .env ファイルを作成します。

```
cp .env.example .env
```

3. テキストエディタで .env ファイルを開きます。ファイル内のサンプル値を独自の設定値に置き換えます。
4. バックエンドの依存関係をインストールします。

```
pip install -r requirements.txt
```

5. バックエンドサーバーを起動します。

```
cd backend  
python oauth_server.py
```

6. 新しいターミナルウィンドウを開き、プロジェクトディレクトリに移動して、フロントエンドサーバーを起動します。

```
cd frontend  
python -m http.server 8001
```

7. ブラウザでアプリケーションを開きます (<http://localhost:8001>)。ブラウザにデモアプリケーションのインターフェイスが表示され、アイデンティティプールの認証をテストする準備が整います。

アイデンティティプールの認証方法を調べる

このセクションでは、Amazon Cognito アイデンティティプールのデモアプリケーションを使用した基本認証フローと拡張認証フローについて説明します。このデモでは、ID プールがさまざまな ID プロバイダーと連携して、アプリケーションユーザーに一時的な AWS 認証情報を提供する方法について説明します。

サンプルアプリケーションの [インタラクティブデモ] セクションで、まずアイデンティティプールでサポートされている 2 種類のアクセスから選択します。

[認証されていない \(ゲスト\) アクセス](#)

まだ認証されていないユーザーに AWS 認証情報を提供します。

認証されたアクセス

ID プロバイダトークンを、使用可能なアクセス許可の全範囲を持つ AWS 認証情報と交換します。 .env ファイルで設定した ID プロバイダーから 1 つを選択します。

認証されていない (ゲスト) アクセス

このステップでは、ID プールのゲストアクセス機能を使用して、認証されていない (ゲスト) ユーザーの一時的な AWS 認証情報を取得する方法を示します。デモアプリケーションでは、拡張フローと基本フローの両方をテストして、ユーザーのサインインを必要とせずに、アイデンティティプールが認証情報を発行する方法を確認します。ゲストアクセスは、認証されたアクセスと同じ API シーケンスを使用しますが、ID プロバイダーのトークン (Google、Facebook、またはエンタープライズプロバイダーからの SAML アサーションなど) からの OAuth トークンなどは提供しません。

認証を必要とせずに AWS への限定的なアクセスをユーザーに提供する方法に関心がある場合は、このまま読み進めてください。ゲストアクセスを実装したら、匿名ユーザーに AWS 認証情報を安全に提供し、2 つの認証フローの違いを理解する方法について説明します。

Important

認証されていないアクセスは、インターネットアクセスを持つすべてのユーザーに認証情報を発行できるため、パブリック APIs やグラフィックアセットなど、最小限のセキュリティを必要とする AWS リソースに最適です。このステップに進む前に、ゲストアクセスを有効にしてアイデンティティプールを設定しているかと、適切な IAM ポリシーを設定してアクセス許可を制限しているかを確認してください。

Guest access with enhanced flow

拡張フローは、2 つの API リクエストを使用して、認証されていないユーザーの AWS 認証情報を取得する合理的なアプローチです。

拡張フローでゲストアクセスをテストするには

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [ゲストアクセス] タブを選択します。
3. [拡張フロー] タブを選択します。
4. [ゲストアクセスをテスト] を選択します。

5. アプリは、追加の認証プロンプトなしで ID プールから一時的な AWS 認証情報を取得します。
6. 認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。
 - a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-07T00:58:21-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。
 - `identityPoolId` を使用した `GetId()` API リクエスト。ゲストアクセスに認証トークンは必要ありません

```
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

有効な場合、ユーザーの `IdentityId` を検索または作成して返します。レスポンスの例は、次のようになります。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

- 返された `identityPoolId` を含む `GetCredentialsForIdentity()`。

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Cognito はゲストアクセスを検証し、で認証されていないロールを内部的に引き受け AWS STS、一時的な AWS 認証情報を返します。(この呼び出しで IAM 認証は行われず、ロールの信頼で `cognito-identity-amazonaws.com` を許可する必要があります)。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-07T00:58:21-07:00"
  }
}
```

Guest access with basic flow

基本フローでは、ID の取得と認証情報の生成に個別の API リクエストを使用して、認証プロセスをきめ細かく制御できます。

基本フローでゲストアクセスをテストするには

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [ゲストアクセス] タブを選択します。
3. [基本フロー] タブを選択します。
4. [ゲストアクセスをテスト] を選択します。
5. アプリは、追加の認証プロンプトなしで ID プールから一時的な AWS 認証情報を取得します。
6. 認証に成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。
 - a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
"SecretAccessKey": "wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
"SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
"Expiration": "2025-08-12T13:36:17-07:00"
}
}
```

b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。

- アイデンティティプール ID を使用した GetId() API リクエスト。ゲストアクセスに認証トークンは必要ありません。

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

有効な場合、ユーザーの IdentityID を検索または作成して返します。レスポンスの例は、次のようになります。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- 返された IdentityID および同じ Logins マップを含む GetOpenIdToken()

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE....."
}
```

このステップで起こること: Amazon Cognito は、cognito-identity.amazonaws.com から、この IdentityId を表す有効期間の短い OpenID Connect ウェブ ID トークンを発行します。トークンには、aud (ID プール ID) や amr (認証済みまたは未認証) など、が

AWS STS 評価する OIDC クレームが含まれます。IAM ロールの信頼ポリシーには、これらのクレームが必要です。

- AssumeRoleWithWebIdentity() - アプリが AWS STS 直接 を呼び出して Amazon Cognito OpenID トークンを一時的な AWS 認証情報と交換する

```
POST sts:AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/
Cognito_IdentityPoolUnauth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE....."
}
```

レスポンス:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "FwoGZXIvYXZzEEXAMPLE....."
  }
}
```

このステップで起こること: 検証されると、一時的な AWS 認証情報を返します。

一時的な認証情報を使用する

これらの一時的な認証情報は標準の AWS 認証情報として機能しますが、ID プールの認証されていない IAM ロールによって定義されたアクセス許可は制限されます。任意の AWS SDK または 使用できます AWS CLI。認証情報を使用して AWS SDKs 「SDK およびツールリファレンスガイド」の「[標準化された認証情報プロバイダー](#)」を参照してください。AWS SDKs

以下の例は完全なリストではありませんが、アイデンティティプールのゲスト機能でユーザーエクスペリエンスを改善できる一般的な方法を示しています。

公開の読み取り専用コンテンツ

以下の例では、Amazon S3 へのゲストユーザーのアクセスを制限するように認証情報プロバイダーを設定します。

Python

```
# Example: Using credentials with boto3
import boto3

# Configure client with temporary credentials
s3_client = boto3.client(
    's3',
    aws_access_key_id='AKIAIOSFODNN7EXAMPLE',
    aws_secret_access_key='wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY',
    aws_session_token='IQoJb3JpZ2luX2VjEEXAMPLE.....'
)

# Make API requests within IAM role permissions
response = s3_client.list_objects_v2(Bucket='my-public-bucket')

# Access public content
for obj in response.get('Contents', []):
    print(f"File: {obj['Key']}, Size: {obj['Size']} bytes")
```

JavaScript

```
// Example: Accessing public content
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

const s3Client = new S3Client({
    region: "us-east-1",
    credentials: {
        accessKeyId: 'AKIAIOSFODNN7EXAMPLE',
        secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY',
        sessionToken: 'IQoJb3JpZ2luX2VjEEXAMPLE.....'
    }
});

// Access public images or documents
const response = await s3Client.send(new GetObjectCommand({
    Bucket: 'my-public-content',
    Key: 'product-catalog.pdf'
}));
```

「ログイン前に試す」機能

以下の例では、ゲストユーザーとして Amazon DynamoDB への読み取り専用アクセスを利用します。

Python

```
# Example: Limited app functionality for trial users
import boto3

dynamodb = boto3.client(
    'dynamodb',
    aws_access_key_id='AKIAIOSFODNN7EXAMPLE',
    aws_secret_access_key='wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY',
    aws_session_token='IQoJb3JpZ2luX2VjEEXAMPLE.....'
)

# Allow guest users to view sample data (limited to 5 items)
response = dynamodb.scan(TableName='SampleProducts', Limit=5)
```

JavaScript

```
// Example: Limited app functionality for trial users
import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-dynamodb";

const dynamodbClient = new DynamoDBClient({
    region: "us-east-1",
    credentials: {
        accessKeyId: 'AKIAIOSFODNN7EXAMPLE',
        secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY',
        sessionToken: 'IQoJb3JpZ2luX2VjEEXAMPLE.....'
    }
});

// Allow guest users to view sample data (limited to 5 items)
const response = await dynamodbClient.send(new ScanCommand({
    TableName: 'SampleProducts',
    Limit: 5
}));
```

ソーシャル ID プロバイダーの認証

このステップでは、Amazon Cognito アイデンティティプールでソーシャル ID プロバイダーを使用するための全体的なフローについて説明します。ソーシャル認証は、ID フェデレーション管理を通じてセキュリティを維持しながら、使い慣れたサインインエクスペリエンスを提供します。Google、Facebook、Amazon などのソーシャル ID プロバイダー (IdP) からサインインし、その IdP トークンを一時的な AWS 認証情報と交換できます。Twitter と Apple の統合も、アイデンティティプールでサポートされていますが、サンプルアプリケーションではサポートされていません。

アイデンティティプール自体は、ユーザーディレクトリではありません。パスワードやプロフィールフィールドは保存されません。代わりに、ユーザーの認証を外部 IdP に委託し、IAM ロールの認証情報を提供することで、認証済みのユーザーが AWS のサービスを直接呼び出すことを承認することに重点を置いています。

Social identity provider with enhanced flow

このセクションでは、ソーシャル ID プロバイダーを使用してユーザーをサインインさせる方法と、拡張フローを使用して Amazon Cognito アイデンティティプールのプロバイダートークンを一時的な認証情報と交換して AWS リソースをリクエストする方法について説明します。

サンプルアプリケーションの拡張フローでソーシャルサインインを使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [拡張フロー] タブを選択します。
4. 設定したサポート対象ソーシャルプロバイダー (Google でサインイン、Facebook でサインイン、Amazon でログインなど) を選択します。
5. サインインして、ユーザーデータをアプリケーションと共有することに同意します。
6. プロバイダーは、アプリケーションのリダイレクト URI にリダイレクトされます。
7. アプリがプロバイダートークンを ID プールに送信し、一時的な AWS 認証情報を取得する
8. アプリケーションは、ウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。
- アプリケーションは、ソーシャル IdP を使用してユーザーをサインインさせ、プロバイダートークンを取得します。アイデンティティプールは、ソーシャルプロバイダーから以下のアーティファクトを受け入れます。

ID プロバイダー	Cognito プロバイダーキー	目的
Google	accounts.google.com	Google Sign-In からの OAuth 2.0 トークン
Facebook	graph.facebook.com	Facebook でログインからのアクセストークン
Amazon	www.amazon.com	Amazon でログインからの OAuth トークン

ソーシャルプロバイダーとの認証が成功すると、アプリケーションはアクセストークンとその他の認証詳細を含む OAuth レスポンスを受け取ります。

```
{
  "access_token": "ya29.A0AS3H6NEXAMPLE.....",
  "expires_in": 3599,
  "scope": "openid https://www.examplesocial....",
  "token_type": "Bearer",
  "id_token": "eyJhbGciOiJIUzI1NiIsEXAMPLE....."
}
```

- アイデンティティプール ID と Logins マップ (ソーシャルプロバイダートークンを含む) を使用した GetId() API リクエスト

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "accounts.google.com": "eyJhbGciOiJIUzI1NiIsEXAMPLE....."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- 返された IdentityID および同じ Logins マップを含む GetCredentialsForIdentity()

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "accounts.google.com": "eyJhbGciOiJIUzI1NiIsEXAMPLE....."
  }
}
```

レスポンス:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-07T00:58:21-07:00"
  },
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

何が起こったか: Amazon Cognito は、設定されたプロバイダーに対してトークンを検証し、プロバイダーの設定に基づいて IAM ロールを選択し、ユーザーに代わって AWS STS を呼び出しました。次に、アイデンティティプールから一時的な認証情報が返されました。

Social identity provider with basic flow

このセクションでは、ソーシャル ID プロバイダーを使用してユーザーにサインインし、基本的なフローを使用して Amazon Cognito ID プールのプロバイダートークンを一時的な認証情報と交換して AWS サービスを呼び出す方法について説明します。

サンプルアプリケーションの基本フローでソーシャルサインインを使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [基本フロー] タブを選択します。
4. 設定したサポート対象ソーシャルプロバイダー (Google でサインイン、Facebook でサインイン、Amazon でログインなど) を選択します。
5. サインインして、ユーザーデータをアプリケーションと共有することに同意します。
6. プロバイダーは、アプリケーションのリダイレクト URI にリダイレクトされます。
7. アプリがプロバイダートークンを ID プールに送信し、一時的な AWS 認証情報を取得する
8. アプリケーションは、ウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。
- アプリケーションは、ソーシャル IdP を使用してユーザーをサインインさせ、プロバイダートークンを取得します。アイデンティティプールは、ソーシャルプロバイダーから以下のアーティファクトを受け入れます。

ID プロバイダー	Cognito プロバイダーキー	目的
Google	accounts.google.com	Google Sign-In からの OAuth 2.0 トークン
Facebook	graph.facebook.com	Facebook でログインからのアクセストークン
Amazon	www.amazon.com	Amazon でログインからの OAuth トークン

ソーシャルプロバイダーとの認証が成功すると、アプリケーションはアクセストークンとその他の認証詳細を含む OAuth レスポンスを受け取ります。

```
{
  "access_token": "ya29.A0AS3H6NEXAMPLE.....",
  "expires_in": 3599,
  "scope": "openid https://www.examplesocial....",
  "token_type": "Bearer",
  "id_token": "eyJhbGciOiJIUzI1NiIsEXAMPLE....."
}
```

- アイデンティティプール ID と Logins マップ (ソーシャルプロバイダートークンを含む) を使用した GetId() API リクエスト

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "accounts.google.com": "token..."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- 返された IdentityID および同じ Logins マップを含む GetOpenIdToken()

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "accounts.google.com": "token..."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE....."
}
```

- OpenID トークンを含む AssumeRoleWithWebIdentity()

```
POST AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/Cognito_IdentityPoolAuth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE....."
}
```

レスポンス:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T14:36:17-07:00"
  }
}
```



```
}
```

起こったこと: Amazon Cognito は、設定されたプロバイダーに対してトークンを検証し、OpenID トークンを発行しました。アプリケーションは AWS STS 直接 を呼び出して IAM ロールを引き受け、一時的な認証情報を受け取ります。

ソーシャルアクセスを理解する

- ソーシャルユーザーは、ソーシャルプロバイダーで認証した後、Amazon Cognito ID プールを通じて一時的な AWS 認証情報を受け取ります。
- 認証された各ユーザーは、セッション間で保持される一意のアイデンティティ ID を取得します。
- これらの認証情報は、認証されたアクセス用に特別に設計された IAM ロールにリンクされ、ゲストアクセスよりも広範なアクセス許可を提供します。
- ソーシャルプロバイダートークンは AWS 認証情報と交換され、ユーザー ID とアクセス許可が維持されます。

Amazon Cognito ユーザープールの認証

このステップでは、ユーザープールの[マネージドログイン](#)統合による Amazon Cognito 認証について説明します。ユーザープールを IdP としてアイデンティティプールにリンクすると、ユーザープールトークンは、アイデンティティプールが一時的な認証情報を発行することを許可します。

User pool authentication with enhanced flow

拡張フローは、単一の API リクエストで Amazon Cognito アイデンティティプールを介して AWS 認証情報を取得するための効率的なアプローチを提供します。

アイデンティティプールの拡張フローで Amazon Cognito ユーザープール認証を使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [拡張フロー] タブを選択します。
4. [Amazon Cognito ユーザープールでサインイン] を選択します。
5. マネージドログインでユーザー名とパスワードを使用してサインインを完了します。
6. ユーザープールは、認可コードをアプリケーションのリダイレクト URI にリダイレクトします。

7. アプリケーションは、ユーザープールの認可コードを JSON ウェブトークンと交換します。
8. アプリケーションは ID トークンを ID プールと一時的な AWS 認証情報と交換します
9. アプリケーションはウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。
 - アプリケーションは、Amazon Cognito でユーザーをサインインさせます。ユーザープールによる認証が成功すると、アプリケーションは ID トークン (JWT) を含む OAuth 2.0 レスポンスを受け取ります。アイデンティティプールは、次のプロバイダーキー形式を使用してユーザープールから JWT ID トークンを受け入れます。

ID プロバイダー	Cognito プロバイダーキー	目的
Amazon Cognito ユーザープール	cognito-idp.{region}.amazonaws.com/{user-pool-id}	Amazon Cognito ユーザープールからの JWT ID トークン

ユーザープールによる認証が成功すると、アプリケーションは ID トークン (JWT) を含む OAuth 2.0 レスポンスを受け取ります。

```
{
  "id_token": "eyJraWQiOiJFWAMPLE.....",
}
```

```
"token_type": "Bearer",  
"expires_in": 3600  
}
```

- identityPoolId と Logins マップ (id_token にマッピングされたユーザープールのプロバイダーキーを含む) を使用した GetId() API リクエスト。Amazon Cognito は、ユーザープール ID トークンの署名、発行者、有効期限、オーディエンス (aud) が、アイデンティティプールでこのユーザープール IdP に登録したアプリケーションクライアント ID の 1 つと一致することを確認しました。

```
POST GetId  
{  
  "AccountId": "111122223333",  
  "IdentityPoolId": "us-east-1:1ac4a76d-1fef-48aa-83af-4224799c0b5c",  
  "Logins": {  
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":  
    "eyJraWQiOiJFWAMPLE....."  
  }  
}
```

有効な場合、ユーザーの IdentityID を検索または作成して返します。レスポンスの例は、次のようになります。

```
{  
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"  
}
```

- 返された identityPoolId および同じ Logins マップ (id_token を含む) を含む GetCredentialsForIdentity()。Amazon Cognito は、ユーザープール ID トークンの署名、発行者、有効期限、オーディエンス (aud) が、アイデンティティプールでこのユーザープール IdP に登録したアプリケーションクライアント ID の 1 つと一致することを再検証します。

```
POST GetCredentialsForIdentity  
{  
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",  
  "Logins": {  
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":  
    "eyJraWQiOiJFWAMPLE....."  
  }  
}
```

```
}
```

有効な場合、IAM ロール (roles-in-token、rules、または default) を選択して、AWS STS を自動的に呼び出し、一時的な AWS 認証情報を返します。レスポンスの例は、次のようになります。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "ASIAW7TIP7EJEXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T14:36:17-07:00"
  }
}
```

User pool authentication with basic flow

基本フローでは、ID の取得と認証情報の生成に個別の API リクエストを使用して、認証プロセスをきめ細かく制御できます。

アイデンティティプールの基本フローで Amazon Cognito ユーザープール認証を使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [基本フロー] タブを選択します。
4. [Amazon Cognito ユーザープールでサインイン] を選択します。
5. マネージドログインでユーザー名とパスワードを使用してサインインを完了します。
6. ユーザープールは、認可コードをアプリケーションのリダイレクト URI にリダイレクトします。
7. アプリケーションは、ユーザープールの認可コードを JSON ウェブトークンと交換します。
8. アプリケーションは ID トークンを ID プールで一時的な AWS 認証情報と交換します。
9. アプリケーションはウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。
- アプリケーションは Amazon Cognito ユーザープールでユーザーをサインインさせ、アーティファクトとして ID トークン (JWT) を取得します。ユーザープールによる認証が成功すると、アプリケーションは ID トークン (JWT) を含む OAuth レスポンスを受け取ります。アイデンティティプールは、このトークンを認証に使用します。

```
{
  "id_token": "eyJraWQiOiJFWAMPLE.....",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

- アイデンティティプール ID と Logins マップ (ユーザープールのプロバイダーキーと ID トークンを値として含む) を使用した GetId() API リクエスト。Amazon Cognito は、ユーザープール ID トークンの署名、有効期限、オーディエンス (aud) が、アイデンティティプールでこのユーザープール IdP に登録したアプリケーションクライアント ID の 1 つと一致することを確認しました。

```
POST GetId
{
  "AccountId": "111122223333",
  "IdentityPoolId": "us-east-1:1ac4a76d-1fef-48aa-83af-4224799c0b5c",
  "Logins": {
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":
    "eyJraWQiOiJFWAMPLE....."
  }
}
```

```
}
```

有効な場合、ユーザーの IdentityID を検索または作成して返します。レスポンスの例は、次のようになります。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- 返された IdentityID および同じ Logins マップを含む GetOpenIdToken()

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":
    "eyJraWQiOiJFWAMPLE....."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE....."
}
```

このステップで起こること: Amazon Cognito は、cognito-identity.amazonaws.com から、この IdentityId を表す有効期間の短い OpenID Connect ウェブ ID トークンを発行します。トークンには、aud (ID プール ID) や amr (認証済みまたは未認証) など、が AWS STS 評価する OIDC クレームが含まれます。IAM ロールの信頼ポリシーには、これらのクレームが必要です。

- AssumeRoleWithWebIdentity() - アプリが AWS STS 直接 を呼び出して Amazon Cognito OpenID トークンを一時的な AWS 認証情報と交換する

```
POST sts:AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/Cognito_IdentityPoolAuth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE.....",
  "RoleSessionName": "CognitoIdentityCredentials"
}
```

```
}
```

レスポンス:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "FwoGZXIvYXdzEEXAMPLE.....",
    "Expiration": "2025-08-12T14:36:17-07:00"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROAW7TIP7EJYEXAMPLE:CognitoIdentityCredentials",
    "Arn": "arn:aws:sts::111122223333:assumed-role/Cognito_IdentityPoolAuth_Role/CognitoIdentityCredentials"
  }
}
```

デモアプリケーションの動作: アプリケーションは OpenID トークンを `GetOpenIdToken()` から AWS STS に送信し、一時的な認証情報を要求しました。AWS STS は検証チェックを実行し、認証情報を発行しました。

ユーザープールへのアクセスを理解する

- ユーザープールのユーザーは、Amazon Cognito ID プールを通じて一時的な AWS 認証情報を受け取ります。
- これらの認証情報は、アイデンティティプール設定で指定した IAM ロールにリンクされています。
- ユーザープール ID トークンは、ID プールを介して AWS 認証情報と交換されます。

SAML 認証

このステップでは、SAML 認証について説明します。ユーザーは、SAML をサポートするエンタープライズ ID プロバイダーでサインインして AWS サービスにアクセスできます。SAML を使用した基本フローは、サンプルアプリケーションではサポートされていません。

SAML authentication with enhanced flow

このセクションでは、SAML ID プロバイダーを使用してユーザーにサインインし、拡張フローを使用して Amazon Cognito ID プールの SAML アサーションを、サービスを呼び出す AWS ための一時的な AWS 認証情報と交換する方法について説明します。

アイデンティティプールの拡張フローで SAML 認証を使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [拡張フロー] タブを選択します。
4. [SAML プロバイダーでサインイン] を選択します。
5. エンタープライズ認証情報を使用してサインインを完了します。
6. ユーザープールは、SAML アサーションをアプリケーションのリダイレクト URI にリダイレクトします。
7. アプリケーションは、ユーザープールの認可コードを JSON ウェブトークンと交換します。
8. アプリケーションは SAML レスポンスを ID プールで一時的な AWS 認証情報と交換します。
9. アプリケーションはウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。

- アプリケーションは、SAML IdP を使用してユーザーをサインインさせ、SAML レスポンスを取得します。アイデンティティプールは、SAML プロバイダー ARN をキーとして使用し、エンタープライズプロバイダーからの SAML アサーションを受け入れます。

ID プロバイダー	Cognito プロバイダーキー	目的
SAML プロバイダー	arn:aws:iam::11112223333:saml-provider/EXAMPLE	エンタープライズ IdP からの SAML アサーション

SAML プロバイダーによる認証が成功すると、アプリケーションは、HTTP POST 経由でコールバック URL への SAML レスポンスを受け取ります。

```
{
  "saml_response": "PD94bWwgdmVyc2lrbj0iMS4wIiBFWAMPLE...",
  "provider_arn": "arn:aws:iam::11112223333:saml-provider/EXAMPLE",
  "status": "Authentication successful"
}
```

- アイデンティティプール ID と Logins マップ (SAML プロバイダー ARN とアサーションを含む) を使用した GetId() API リクエスト

```
POST GetId
{
  "AccountId": "11112223333",
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE1111",
  "Logins": {
    "arn:aws:iam::11112223333:saml-provider/EXAMPLE":
    "PD94bWwgdmVyc2lrbj0iMS4wIiBFWAMPLE..."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE2222"
}
```

- 返された IdentityID および同じ Logins マップを含む GetCredentialsForIdentity()。

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "arn:aws:iam::111122223333:saml-provider/EXAMPLE":
    "PD94bWwgdmVyc2lrbj0iMS4wIiBFWAMPLE..."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE....."
  }
}
```

何が起きたか: Amazon Cognito は、設定されたプロバイダーに対して SAML アサーションを検証し、SAML 属性またはルールに基づいて IAM ロールを選択し、AWS STS ユーザーに代わって呼び出しました。

SAML へのアクセスを理解する

- エンタープライズユーザーは、SAML プロバイダーで認証した後、Amazon Cognito ID プールから一時的な AWS 認証情報を受け取ります。
- 認証された各ユーザーは、セッション間で保持される一意のアイデンティティ ID を取得します。
- これらの認証情報は、認証されたアクセス用に特別に設計された IAM ロールにリンクされ、ゲストアクセスよりも広範なアクセス許可を提供します。
- SAML アサーションは AWS 認証情報と交換され、ユーザー ID とエンタープライズ属性が維持されます。

OpenID Connect (OIDC) 認証

このステップでは、エンタープライズ ID プロバイダーによる OIDC 認証について説明します。ユーザーは、組織のエンタープライズ ID プロバイダー (Azure AD、Okta、Google Workspace など) を通じてサインインして、AWS サービスにアクセスできます。標準ベースの認証と AWS リソースの統合に関心がある場合は、このまま読み進めてください。OIDC 認証を実装したら、OIDC クレームを活用してきめ細かなアクセスコントロールを行う方法について説明します。

OIDC authentication with enhanced flow

このセクションでは、OIDC ID プロバイダーを使用してユーザーにサインインし、拡張フローを使用して Amazon Cognito ID プールの OIDC トークンを一時的な AWS 認証情報と交換して サービスを呼び出す方法を示します。

アイデンティティプールの拡張フローで OIDC サインインを使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [拡張フロー] タブを選択します。
4. [OIDC プロバイダーへのサインイン] を選択します。
5. エンタープライズ認証情報を使用してサインインを完了します。
6. OIDC プロバイダーは、認可コードをアプリケーションにリダイレクトします。
7. アプリケーションは、ユーザープールの認可コードを JSON ウェブトークンと交換します。
8. アプリケーションは OIDC トークンを ID プールに送信し、一時的な AWS 認証情報を取得します。
9. アプリケーションはウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
```

```

    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}

```

b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。

- アプリケーションは OIDC IdP を使用してユーザーをサインインさせ、ID トークンを取得します。アイデンティティプールは、エンタープライズプロバイダーからの OIDC トークンを受け入れます。

ID プロバイダー	Cognito プロバイダーキー	目的
OIDC プロバイダー	example-provider.com/oauth2/default	エンタープライズ IdP からの OIDC ID トークン

OIDC プロバイダーによる認証が成功すると、アプリケーションはトークンを含む OAuth 2.0 レスポンスを受け取ります。

```

{
  "token_type": "Bearer",
  "expires_in": 3600,
  "access_token": "eyJraWQiOiJFWAMPLE.....",
  "scope": "email openid profile",
  "id_token": "eyJraWQiOiJFWAMPLE....."
}

```

- アイデンティティプール ID と Logins マップ (OIDC プロバイダートークンを含む) を使用した GetId() API リクエスト

```

POST GetId
{
  "AccountId": "111122223333",
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE....."
  }
}

```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- 返された IdentityID および同じ Logins マップを含む
GetCredentialsForIdentity()

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE....."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE....."
  }
}
```

何が起きたか: Amazon Cognito は、設定されたプロバイダーに対して OIDC トークンを検証し、IAM ロール (デフォルト、クレームベース、またはルールマップ) を選択し、AWS STS ユーザーに代わって を呼び出しました。

OIDC authentication with basic flow

このセクションでは、OIDC ID プロバイダーを使用してユーザーにサインインし、基本フローを使用して Amazon Cognito ID プールの OIDC トークンを一時的な AWS 認証情報と交換して サービスを呼び AWS 出す方法を示します。

アイデンティティプールの基本フローで OIDC サインインを使用する

1. デモアプリケーションで、[インタラクティブデモ] セクションに移動します。
2. [認証されたアクセス] タブを選択します。
3. [基本フロー] タブを選択します。
4. [OIDC プロバイダーへのサインイン] を選択します。
5. エンタープライズ認証情報を使用してサインインを完了します。
6. OIDC プロバイダーは、認可コードをアプリケーションにリダイレクトします。
7. アプリケーションは、ユーザープールの認可コードを JSON ウェブトークンと交換します。
8. アプリケーションは OIDC トークンを ID プールに送信し、一時的な AWS 認証情報を取得します。
9. アプリケーションはウェブインターフェイスに [結果] パネルを表示します。

認証が成功すると、ウェブインターフェイスに [結果] パネルが表示されます。結果は、2 つのオプションを使用して参照できます。

- a. 認証情報のみの表示ボタン: API フローの詳細なしで生成された一時的な AWS 認証情報を直接表示する場合は、このボタンを選択します。

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE.....",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

- b. [詳細な API フローを表示] ボタン: API リクエストをステップバイステップで確認する場合は、このボタンを選択します。
 - アプリケーションは OIDC IdP を使用してユーザーをサインインさせ、ID トークンを取得します。アイデンティティプールは、エンタープライズプロバイダーからの OIDC トークンを受け入れます。

ID プロバイダー	Cognito プロバイダーキー	目的
OIDC プロバイダー	example-provider.com/oauth2/default	エンタープライズ IdP からの OIDC ID トークン

OIDC プロバイダーによる認証が成功すると、アプリケーションはトークンを含む OAuth 2.0 レスポンスを受け取ります。

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "access_token": "eyJraWQiOiJFWAMPLE.....",
  "scope": "openid email profile",
  "id_token": "eyJraWQiOiJFWAMPLE....."
}
```

- アイデンティティプール ID と Logins マップ (OIDC プロバイダートークンを含む) を使用した GetId() API リクエスト

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE....."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- 返された IdentityId および同じ Logins マップを含む GetOpenIdToken()。

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
```

```
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE....."
  }
}
```

レスポンス:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE....."
}
```

- OpenID トークンを含む AssumeRoleWithWebIdentity()

```
POST AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/Cognito_IdentityPoolAuth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE....."
}
```

レスポンス:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "FwoGZXIvYXdzEEXAMPLE.....",
    "Expiration": "2025-08-12T14:36:17-07:00"
  }
}
```

起こったこと: Amazon Cognito は、設定されたプロバイダーに対して OIDC トークンを検証し、OpenID トークンを返しました。アプリケーションは AWS STS 直接 を呼び出して適切な IAM ロールを引き受け、有効期間の短い認証情報を受け取りました。

OIDC 認証を理解する

- 標準ベース: OIDC は OAuth 2.0 上に構築され、標準化された ID 情報を提供します。
- トークンの検証: ID トークンの信頼性を検証できます。

- クレームベースのアクセス: OIDC クレームは、ロールマッピングとアクセスコントロールに使用できます。
- エンタープライズ統合: 一般的なエンタープライズ ID プロバイダーと連携します。

次の手順

デモアプリケーションをセットアップして確認したら、以下のことができます。

- まだテストしていない追加の ID プロバイダーを設定する
- 拡張認証と基本認証の両方を試して、その違いを理解する
- 独自のユースケースに合わせてデモをカスタマイズする
- Amazon Cognito アイデンティティプールを独自のアプリケーションに統合する

Amazon Cognito のガイド付きセットアップオプション

構造化済みの、ガイド付きエクスペリエンスで Amazon Cognito の機能を評価することをお勧めします。ユーザープールとアイデンティティプールを用いてカスタマイズされたエクスペリエンスを提供する外部リソースをいくつか紹介します。

ワークショップに参加する

AWS Workshop Studio では、Amazon Cognito の大部分の機能のセットアップについて説明する [ワークショップを開催](#) しています。これらの機能には、ユーザープール API、ユーザープールのホストされた UI、アイデンティティプール、セキュリティ設定が含まれます。

例からアプリケーションコードを追加する

このガイドの [コード例](#) の章には、ユーザープールとアイデンティティプールで使えるアプリケーションコードが含まれています。コード例の章のユーザープールセクションには、個々のオペレーションをカバーする短いスニペットと、さまざまなプログラミング言語のエンドツーエンドのサンプルアプリケーションについてのやや長い例が含まれています。

AWS Amplify でフルスタックアプリケーションを作成する

[AWS Amplify](#) は、アプリケーションとユーザーインターフェイスの開発とホストを行うデベロッパー向けの AWS のサービスです。Amazon Cognito は Amplify の認証コンポーネントです。アプリケーションに認証を追加すると、Amplify は Amazon Cognito ユーザープールとアイデンティティプールリソースのデプロイを自動化できます。また、[Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#) も参照してください。

GitHub に関するその他の Amazon Cognito アプリケーションリソース

- [Amazon Cognito 用の .NET を使用した認証フローの例](#)
- [Amazon Cognito パスワードレス認証](#)
- [Amazon Verified Permissions を使用した PetStore の例](#)
- [ABAC + アイデンティティプールを使用して AWS リソースにアクセスするサンプル React アプリケーション](#)
- [AWS CDK を使用した Amazon Cognito および API Gateway ベースの Machine to Machine 認可](#)
- [Amazon Cognito、API Gateway、IAM を使用したきめ細かな認可の構築](#)
- [CloudFront authorization@edge](#)

その他のワークショップ

- [Amazon Cognito と WebAuthn によるパスワードレス認証の実装](#)
- [Amazon Cognito ワークショップ](#)
- [Amazon Cognito トラブルシューティングワークショップ](#)
- [Amazon Cognito と Verified Permissions による認証と認可](#)
- [Amazon Cognito JWT の詳細](#)

ブログ投稿

- [Amazon CloudFront プロキシを使用して Amazon Cognito のパブリッククライアントを保護する](#)
- [Azure AD を使用してフェデレーション認証用に Amazon Cognito をセットアップする方法](#)
- [ウェブアプリケーション認証の簡素化: Amazon Cognito ユーザープールを使用した AD FS フェデレーションのガイド](#)

Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する

Amazon Cognito の実装は、AWS マネジメントコンソール または AWS SDK 管理ツールと、アプリケーション内の SDK ライブラリを組み合わせたものです。Amazon Cognito コンソールは、Amazon Cognito ユーザープールとアイデンティティプールを設定および管理するためのビジュアルインターフェイスです。

Amazon Cognito ユーザープールと最も簡単に統合するには、[マネージドログイン](#)を使用します。マネージドログインは、Amazon Cognito ユーザープールを迅速にテストおよびデプロイするための、即座に使用できるウェブベースのサインインアプリケーションです。マネージドログインによるユーザープール認証には、ホストされたサインインページにユーザーを誘導する OpenID Connect (OIDC) ライブラリが必要です。この一連のユーザーインタラクティブなリダイレクトウェブエンドポイントでは、サードパーティによるサインイン、多要素認証 (MFA)、認証フローの選択などの認証フローを Amazon Cognito が処理します。アプリケーションは、Amazon Cognito がレスポンスで返す認証結果だけを処理する必要があります。

また、アプリケーションに AWS SDK を追加したり、認証インターフェイスをカスタムビルドしたり、ユーザーの認証と認可のために API オペレーションを呼び出すこともできます。[AWS Amplify](#)は、バックエンドで Amazon Cognito 認証を使用してフルスタックアプリケーションを構築 AWS のサービス するための です。

例えば、アプリケーションは、ユーザーのサインインのためにマネージドログインを呼び出し、次にアプリケーションコードからトークンエンドポイントを呼び出して、ユーザーの認可コードをトークンと交換する場合があります。この場合、アプリケーションはユーザーのトークンを解釈して保存し、認証と認可のための適切なコンテキストでトークンを提示する必要があります。Amplify は、これらのプロセス用の組み込み機能を備えたガイド付き統合ツールを追加します。

Amazon Cognito リソースをすべてコードで構築することもできます。ID プールには、ユーザープールと同じマネージド認証オプションはありません。アプリケーションの AWS 認証情報にアクセスするには、インポートされた SDK モジュールに ID プールオペレーションを実装します。独自にカスタム構築したアプリケーションコードの使用を開始するには、[AWS SDK](#) の Amazon Cognito [コード例](#)を参照してください。OpenID Connect ID プロバイダーとして Amazon Cognito と統合するには、[OpenID Connect デベロッパーツール](#)を使用します。

Amazon Cognito の認証と認可を使用する前に、アプリプラットフォームを選択し、サービスと統合するためのコードを準備します。AWS SDKs「」を参照してください[AWS SDKsによる認](#)

[証](#)。AWS CLI は Amazon Cognito およびその他のコマンドライン SDK であり AWS のサービス、Amazon Cognito API オペレーションとその構文に慣れるための貴重な場所です。

Note

Amazon Cognito の一部のコンポーネントは、API でのみ設定できます。例えば、ユーザープールの[カスタム SMS または E メール送信者](#)の Lambda トリガーは、[CreateUserPool](#) または [UpdateUserPool](#) API リクエスト内の [UserPool](#) クラスの LambdaConfig プロパティを更新するリクエストでのみ設定できます。

Amazon Cognito ユーザープール API は、その名前空間を複数の API オペレーションクラスと共有します。1 つのクラスでは、ユーザープールとそのプロセス、ID プロバイダー、ユーザーを設定します。別のクラスでは、パブリッククライアントでユーザーがサインイン、サインアウト、プロフィールの管理を行うための認証されていないオペレーションを提供します。API オペレーションの最後のクラスでは、機密のサーバー側クライアントで、独自の AWS 認証情報で認可したユーザーオペレーションを実行します。アプリケーションコードの実装を始める前に、目的のアプリケーションアーキテクチャを知っておく必要があります。詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

トピック

- [による認証 AWS Amplify](#)
- [AWS SDKsによる認証](#)
- [Amazon Cognito による認証の仕組み](#)
- [AWS SDK でのこのサービスの使用](#)
- [Amazon Verified Permissions による承認](#)

による認証 AWS Amplify

AWS Amplify は、ウェブおよびモバイルアプリケーションを構築するための完全なソリューションです。Amplify を使用すると、Amplify ライブラリを使用して既存のリソースに接続したり、Amplify コマンドラインインターフェイス (CLI) を使用して新しいリソースを作成および設定したりできます。Amplify には、[Authenticator](#) などの UI コンポーネントも接続されており、アプリケーションでのサインインやサインアップエクスペリエンスの設定とカスタマイズに使用できます。

フロントエンドアプリケーションで Amplify の認証機能を使用するには、プラットフォーム別の以下のドキュメントを参照してください。

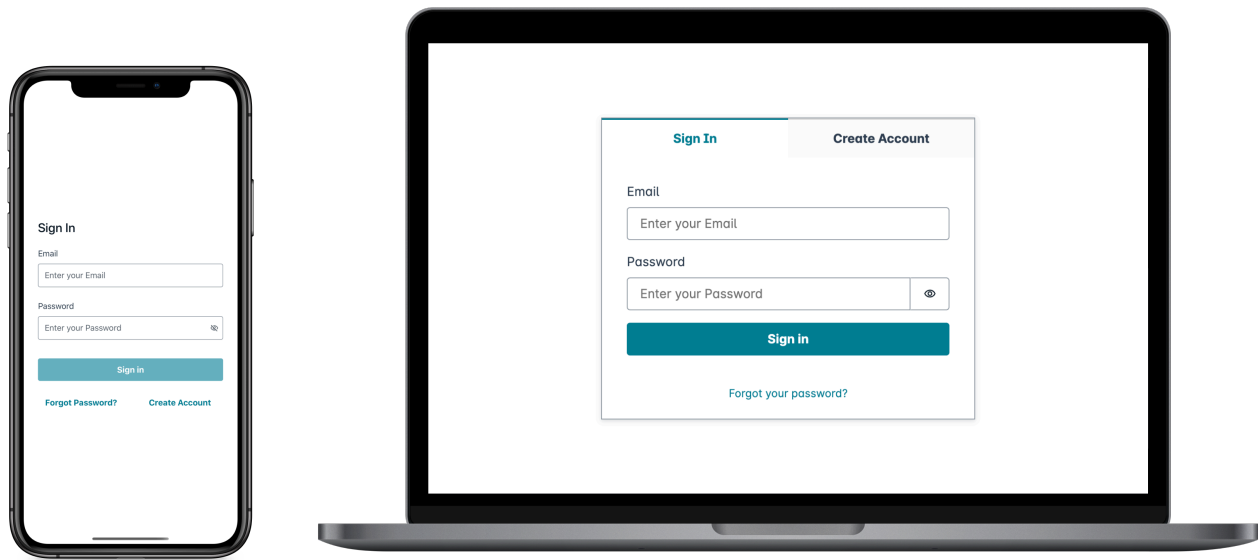
- [React 向けの Amplify 認証](#)
- [React Native 向けの Amplify 認証](#)
- [Swift 向けの Amplify 認証 \(iOS\)](#)
- [Android 向けの Amplify 認証](#)
- [Flutter 向けの Amplify 認証](#)

Amplify ライブラリはオープンソースで、[GitHub](#) で利用できます。Amplify Auth が Amazon Cognito 認証をどのように実装するかの詳細については、以下のライブラリを参照してください。

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

Amplify によるユーザーインターフェイス (UI) の作成

[ユーザープールのマネージドログイン](#)は、ウェブまたはモバイルアプリの認証フロントエンドの最も重要なニーズを満たすことができます。マネージドログインで対応できるパラメータを超えてユーザーインターフェイス (UI) をカスタマイズするには、アプリケーションをカスタム構築します。[Amplify UI](#) は、さまざまな言語でのカスタマイズ可能なフロントエンドコンポーネントのコレクションです。



カスタム認証コンポーネントの使用を開始するには、Authenticator コンポーネントに関する以下のドキュメントを参照してください。

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

AWS SDKsによる認証

安全なバックエンドを使用して Amazon Cognito とやり取りする独自の ID マイクロサービスを構築するには、任意の言語で AWS SDK を使用して Amazon Cognito ユーザープールと Amazon Cognito ID プール API に接続します。

各 API オペレーションの詳細については、「[Amazon Cognito ユーザープール API リファレンス](#)」と「[Amazon Cognito API リファレンス](#)」を参照してください。これらのドキュメントには、サポート

されているプラットフォームでさまざまな SDK を使用するためのリソースに関する、「[以下の資料も参照してください](#)」セクションが含まれています。

- [AWS コマンドラインインターフェイス](#)
- [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](#)

Amazon Cognito による認証の仕組み

顧客が Amazon Cognito ユーザープールにサインインすると、アプリケーションは JSON ウェブ トークン (JWT) を受け取ります。

顧客がユーザープールトークンまたは他のプロバイダーを使用して ID プールにサインインすると、アプリケーションは一時的な AWS 認証情報を受け取ります。

ユーザープールサインインを使用すると、AWS SDK を使用して認証と認可を完全に実装できます。独自のユーザーインターフェイス (UI) コンポーネントを構築しない場合は、構築済みのウェブ UI (マネージドログイン) またはサードパーティの ID プロバイダー (IdP) のサインインページを起動できます。

このトピックでは、アプリケーションが Amazon Cognito とやり取りして ID トークンによる認証、アクセストークンによる認可、および ID プール認証情報 AWS のサービス によるアクセスを行う方法の概要を説明します。

トピック

- [マネージドログインによるユーザープール認証](#)
- [AWS SDK を使用したユーザープール API の認証と認可](#)
- [サードパーティー ID プロバイダーによるユーザープール認証](#)
- [アイデンティティプール認証](#)

マネージドログインによるユーザープール認証

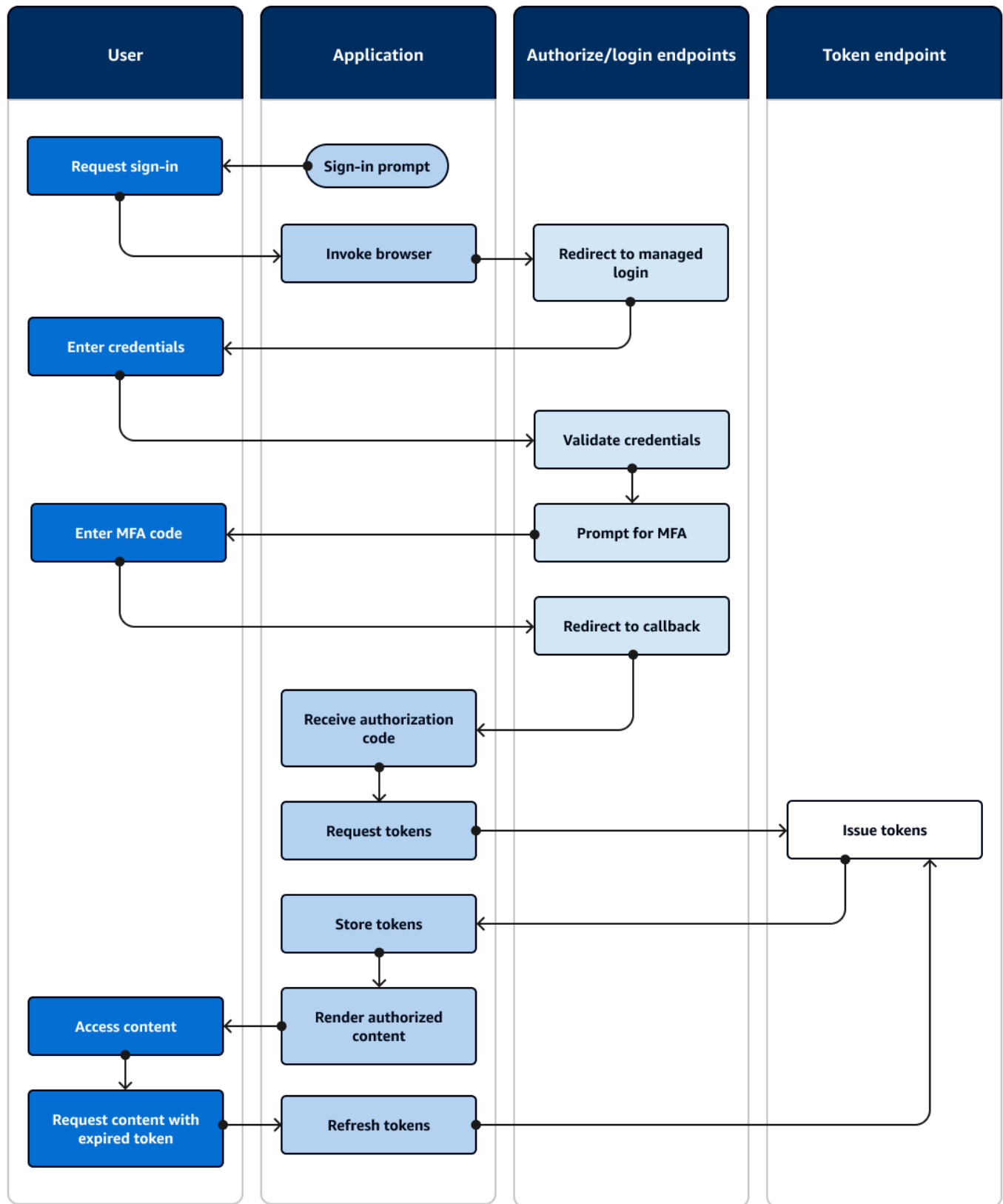
[マネージドログイン](#)は、ユーザープールとアプリケーションクライアントにリンクされたウェブサイトです。ユーザーのサインイン、サインアップ、パスワードリセットの操作を実行できます。認証用のマネージドログインコンポーネントを持つアプリケーションでは、デベロッパーによる実装の労力を軽減できます。アプリケーションは、認証用の UI コンポーネントをスキップし、マネージドログインのウェブページをユーザーのブラウザで起動できます。

アプリケーションは、ウェブまたはアプリケーションのリダイレクトロケーションを使用してユーザーの JWT を収集します。マネージドログインを実装したアプリケーションは、OpenID Connect (OIDC) の IdP であるかのように、認証のためにユーザープールに接続できます。

マネージドログインは、アプリケーションが OIDC 認可サーバーの認証サービスを必要とするが、カスタム認証、アイデンティティプールの統合、ユーザー属性のセルフサービスなどの機能をすぐには必要としないモデルに適しています。これらの高度なオプションの一部を使用する場合は、SDK のユーザープールコンポーネントで実装できます。

OIDC 実装に主に依存する、マネージドログインやサードパーティ IdP の認可モデルは、OAuth 2.0 スコープを使用する高度な認可モデルに最適です。

次の図は、マネージドログイン認証の一般的なサインインセッションを示しています。



マネージドログイン認証フロー

1. ユーザーがアプリケーションにアクセスします。
2. [サインイン] リンクを選択します。
3. アプリケーションは、ユーザープールドメインのマネージドログインページのサインインプロンプトへと、ユーザーを誘導します。
4. ユーザー名とパスワードを入力します。
5. ユーザープールはユーザーの認証情報を検証し、ユーザーが多要素認証 (MFA) をアクティブ化したと判断します。
6. マネージドログインページは、MFA コードを入力するようユーザーに求めます。
7. ユーザーが MFA コードを入力します。
8. ユーザープールは、ユーザーをアプリケーション URL にリダイレクトします。
9. アプリケーションは、マネージドログインで[コールバック URL](#) に追加した URL リクエストパラメータから認証コードを収集します。
10. アプリケーションは、認証コードを使用してトークンをリクエストします。
11. トークンエンドポイントは JWT をアプリケーションに返します。
12. アプリケーションは、ユーザーの JWT についてデコード、検証、保存、キャッシングを行います。
13. アプリケーションには、リクエストの対象である、アクセスコントロールコンポーネントが表示されます。
14. ユーザーはコンテンツを表示します。
15. その後、ユーザーのアクセストークンの有効期限が切れ、アクセスコントロールコンポーネントの表示をリクエストします。
16. アプリケーションは、ユーザーのセッションを持続させる必要があると判断します。更新トークンを使用して、トークンエンドポイントから新しいトークンをリクエストします。

バリエーションとカスタマイズ

[ブランディングエディタ](#)を使用して、ユーザープール全体または任意の[アプリケーションクライアント](#)レベルで、マネージドログインページの外観と操作感をカスタマイズできます。また、独自の ID プロバイダー、スコープ、ユーザー属性へのアクセス、高度なセキュリティ設定を使用して[アプリケーションクライアントを設定](#)することもできます。

関連リソース

- [ユーザープールのマネージドログイン](#)
- [スコープ、M2M、リソースサーバー](#)
- [ユーザープールのエンドポイントとマネージドログインのリファレンス](#)

AWS SDK を使用したユーザープール API の認証と認可

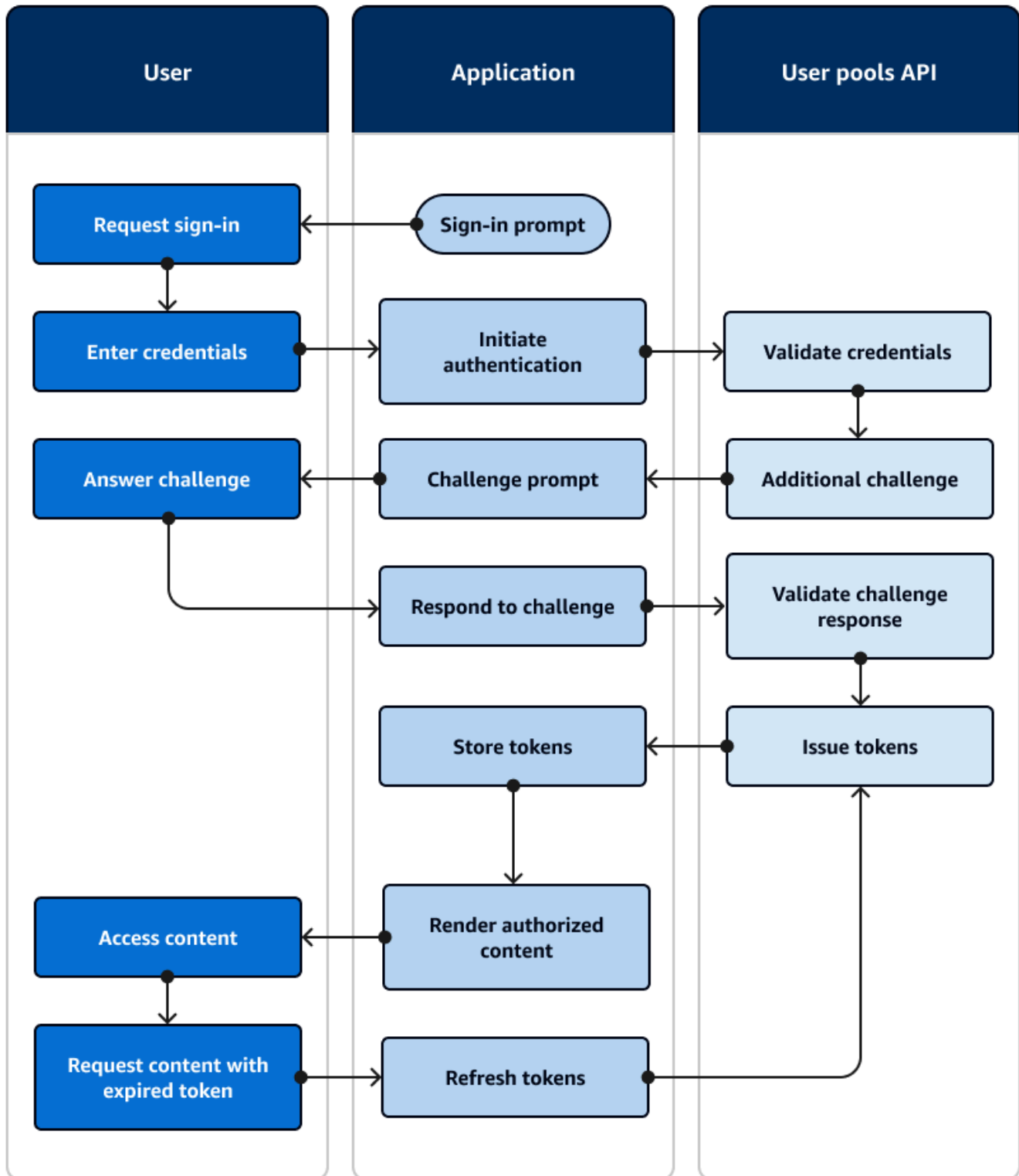
AWS は、[さまざまな開発者フレームワーク](#)で Amazon Cognito ユーザープールまたは Amazon Cognito ID プロバイダー用のコンポーネントを開発しました。これらの SDK に組み込まれているメソッドは、[Amazon Cognito ユーザープール API](#) を呼び出します。同じユーザープール API 名前空間には、ユーザープールの設定とユーザー認証のためのオペレーションがあります。詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

API 認証は、アプリケーションに既存の UI コンポーネントがあって、かつ、主にユーザーディレクトリとしてユーザープールに依存するモデルに適合します。この設計では、Amazon Cognito を大型アプリケーション内のコンポーネントとして追加します。これには、複雑なチャレンジとレスポンスの連鎖を処理するためのプログラムロジックが必要です。

このアプリケーションでは、OpenID Connect (OIDC) の依拠しているパーティーによる完全な実装を実装する必要はありません。代わりに、JWT をデコードして使用できます。[ローカルユーザー](#)のユーザープール機能一式にアクセスするには、開発環境で Amazon Cognito SDK を使用して認証を構築します。

カスタム OAuth スコープを使用した API 認証は、外部 API 認証に向いていません。API 認証からアクセストークンにカスタムスコープを追加するには、[トークン生成前の Lambda トリガー](#) を使用して実行時にトークンを変更します。

次の図は、API 認証の一般的なサインインセッションを示しています。



API 認証フロー

1. ユーザーがアプリケーションにアクセスします。
2. [サインイン] リンクを選択します。
3. ユーザー名とパスワードを入力します。
4. アプリケーションは、[InitiateAuth](#) リクエストを行うメソッドを呼び出します。リクエストはユーザーの認証情報をユーザープールに渡します。
5. ユーザープールはユーザーの認証情報を検証し、ユーザーが多要素認証 (MFA) をアクティブ化したと判断します。
6. ユーザープールは、MFA コードをリクエストするチャレンジで応答します。
7. アプリケーションは、ユーザーから MFA コードを収集するプロンプトを生成します。
8. アプリケーションは、[RespondToAuthChallenge](#) API リクエストを行うメソッドを呼び出します。リクエストはユーザーの MFA コードを渡します。
9. ユーザープールは、ユーザーの MFA コードを検証します。
10. ユーザープールは、ユーザーの JWT で応答します。
11. アプリケーションは、ユーザーの JWT についてデコード、検証、保存、キャッシングを行います。
12. アプリケーションには、リクエストの対象である、アクセスコントロールコンポーネントが表示されます。
13. ユーザーはコンテンツを表示します。
14. その後、ユーザーのアクセストークンの有効期限が切れ、アクセスコントロールコンポーネントの表示をリクエストします。
15. アプリケーションは、ユーザーのセッションを持続させる必要があると判断します。更新トークンを使用して [InitiateAuth](#) メソッドを再度呼び出し、新しいトークンを取得します。

バリエーションとカスタマイズ

このフローは、独自のカスタム認証チャレンジなど、追加のチャレンジで拡張できます。パスワードが漏洩したユーザー、または予期しないサインイン特性によって悪意のあるサインインの試みがなされたことを示している可能性があるユーザーのアクセスを自動的に制限できます。このフローは、オペレーションがサインアップ、ユーザー属性の更新、パスワードのリセットを行う場合とほぼ同じです。これらのフローのほとんどには、パブリック (クライアント側) と機密 (サーバー側) の API オペレーションが重複しています。

関連リソース

- [Amazon Cognito ユーザープール API](#)
- [ユーザープールの開始方法](#)
- [Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#)
- [API、OIDC、マネージドログインページの認証についての理解](#)

サードパーティー ID プロバイダーによるユーザープール認証

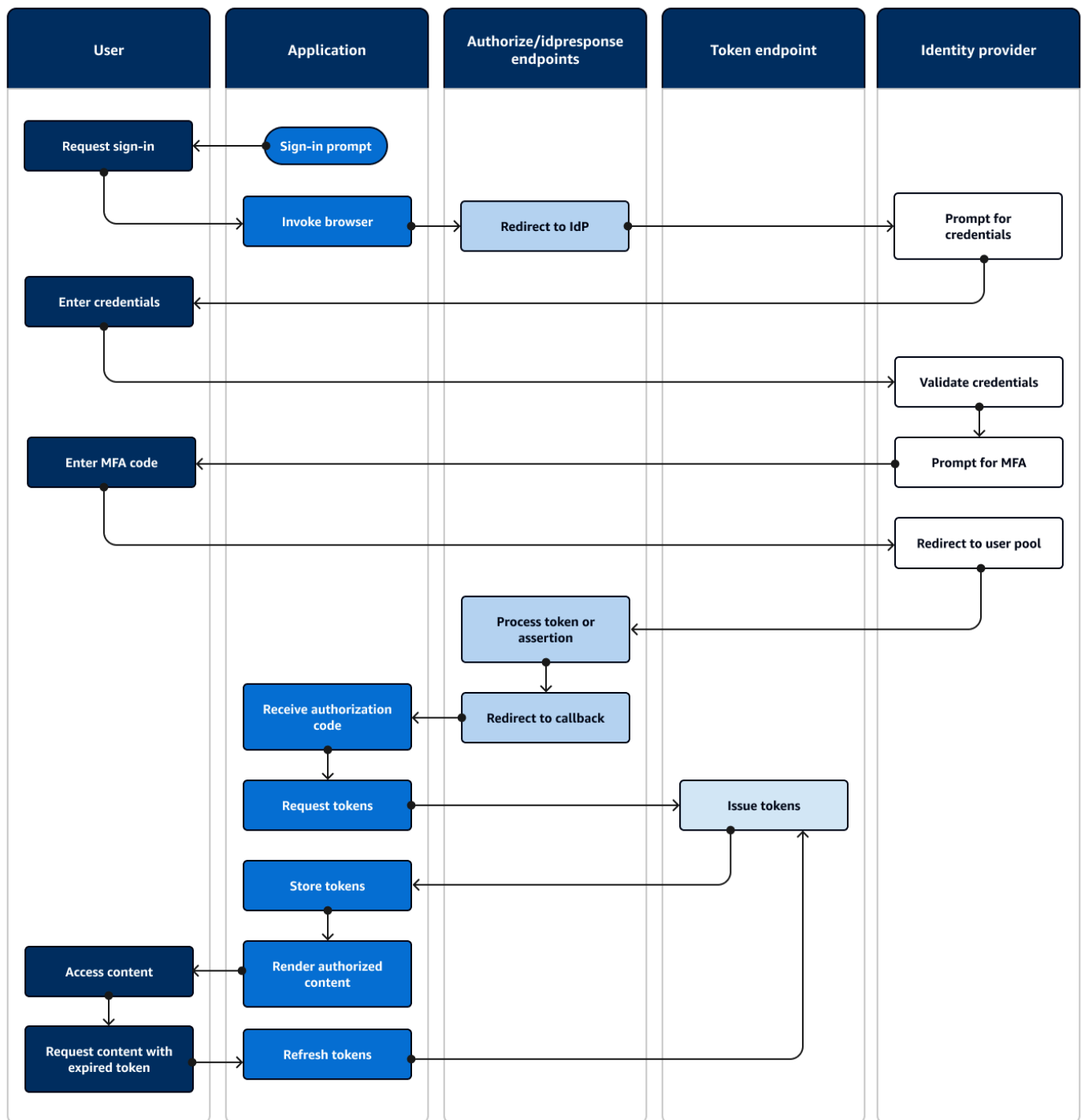
外部 ID プロバイダー (IdP) によるサインイン (またはフェデレーション認証) は、[マネージドログイン](#)と似たモデルです。アプリケーションはユーザープールへの OIDC 依拠している当事者であり、ユーザープールは IdP へのパススルーとして機能します。IdP は、Facebook や Google などのコンシューマーユーザーディレクトリにすることも、SAML 2.0 や Azure などの OIDC エンタープライズディレクトリにすることもできます。

アプリケーションは、ユーザーのブラウザでマネージドログインを呼び出す代わりに、ユーザープールの[認可サーバー](#)でリダイレクトエンドポイントを呼び出します。ユーザーのビューから、アプリケーションのサインインボタンを選択します。次に、IdP からサインインするよう求められます。マネージドログイン認証と同じように、アプリケーションはアプリ内のリダイレクト場所で JWT を収集します。

サードパーティー IdP による認証は、アプリケーションにサインアップするときにユーザーが新しいパスワードを作らないモデルに適合します。サードパーティ認証は、マネージドログイン認証を実装したアプリケーションに簡単に追加できます。実際、マネージドログインとサードパーティー IdP は、ユーザーのブラウザで何を呼び出すかのわずかな違いに基づいて、一貫した認証結果を生成します。

マネージドログイン認証と同様に、フェデレーション認証は、OAuth 2.0 スコープを使用する高度な認可モデルに最適です。

次の図は、フェデレーション認証の一般的なサインインセッションを示しています。



フェデレーション認証フロー

1. ユーザーがアプリケーションにアクセスします。
2. [サインイン] リンクを選択します。

3. アプリケーションは、IdP を使用してサインインプロンプトをユーザーに指示します。
4. ユーザー名とパスワードを入力します。
5. IdP はユーザーの認証情報を検証し、ユーザーが多要素認証 (MFA) をアクティブ化したと判断します。
6. IdP は、ユーザーに MFA コードを入力するように求めます。
7. ユーザーが MFA コードを入力します。
8. IdP は、SAML レスポンスまたは認証コードを使用してユーザーをユーザープールにリダイレクトします。
9. ユーザーが認証コードを渡した場合、ユーザープールはコードを IdP トークンにサイレント交換します。ユーザープールは IdP トークンを検証し、新しい認証コードを使用してユーザーをアプリケーションにリダイレクトします。
10. アプリケーションは、[コールバック URL](#) にユーザープールが追加した URL リクエストパラメータから認証コードを収集します。
11. アプリケーションは、認証コードを使用してトークンをリクエストします。
12. トークンエンドポイントは JWT をアプリケーションに返します。
13. アプリケーションは、ユーザーの JWT についてデコード、検証、保存、キャッシングを行います。
14. アプリケーションには、リクエストの対象である、アクセスコントロールコンポーネントが表示されます。
15. ユーザーはコンテンツを表示します。
16. その後、ユーザーのアクセストークンの有効期限が切れ、アクセスコントロールコンポーネントの表示をリクエストします。
17. アプリケーションは、ユーザーのセッションを持続させる必要があると判断します。更新トークンを使用して、トークンエンドポイントから新しいトークンをリクエストします。

バリエーションとカスタマイズ

[マネージドログイン](#)でフェデレーション認証を開始できます。ここで、ユーザーは、[アプリケーションクライアント](#)に割り当てられたリストから IdP を選択できます。マネージドログインは、E メールアドレスを入力するよう求め、対応する SAML IdP に[ユーザーのリクエストを自動的にルーティング](#)することもできます。サードパーティ ID プロバイダーによる認証では、マネージドログインとのユーザーインタラクションは必要ありません。アプリケーションは、ユーザーの[認可サーバーリクエスト](#)にリクエストパラメータを追加し、ユーザーに IdP サインインページにサイレントリダイレクトさせることができます。

関連リソース

- [サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン](#)
- [スコープ、M2M、リソースサーバー](#)
- [ユーザープールのエンドポイントとマネージドログインのリファレンス](#)

アイデンティティプール認証

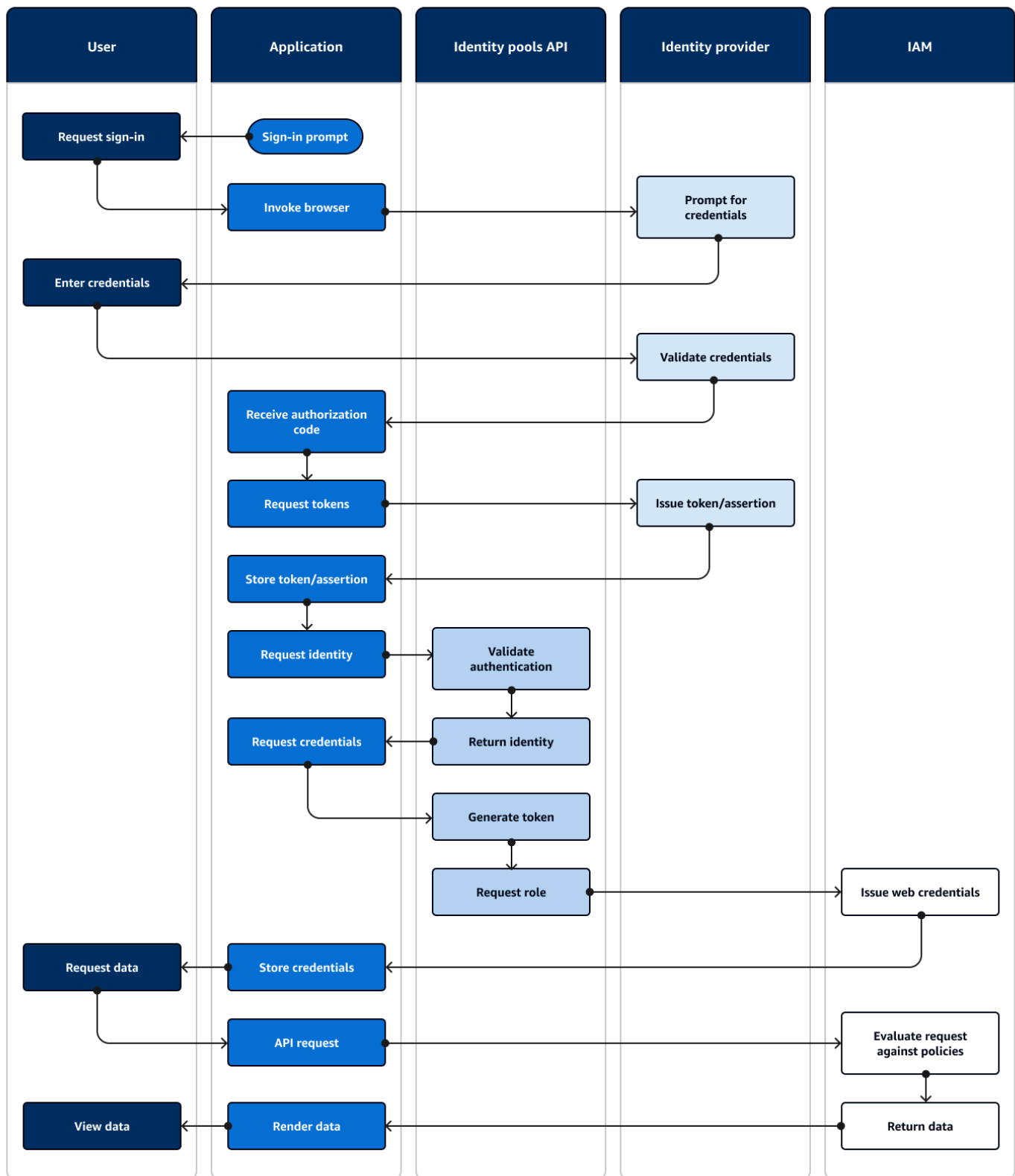
アイデンティティプールは、関数、API 名前空間、および SDK モデルのユーザープールとは異なる、アプリケーションのコンポーネントです。ユーザープールがトークンベースの認証と認可を提供する場合、ID プールは AWS Identity and Access Management (IAM) の認可を提供します。

アイデンティティプールに IdP のセットを割り当て、それを使用してユーザーにサインインできます。ユーザープールはアイデンティティプール IdP として緊密に統合されており、アイデンティティプールに対して、アクセスコントロールの大半のオプションを提供します。同時に、アイデンティティプールにはさまざまな認証オプションがあります。ユーザープールは、SAML、OIDC、ソーシャル、デベロッパー、ゲスト ID ソースを、ID プールからの一時的な AWS 認証情報へのルートとして結合します。

アイデンティティプールによる認証は外部です。以前に説明したユーザープールフローのいずれかに従うか、別の IdP で独自に開発するフローに従います。アプリケーションが初期認証を実行すると、検証をアイデンティティプールに渡して、その見返りとして一時的なセッションを受け取ります。

ID プールを使用した認証は、IAM 認可 AWS のサービスを使用して のアプリケーションアセットとデータのアクセスコントロールを適用するモデルに適合します。[ユーザープールの API 認証](#)と同様に、成功したアプリケーションには、ユーザーの利益のためにアクセスする各サービスの AWS SDKs が含まれます。AWS SDKs、ID プール認証からの認証情報を API リクエストの署名として適用します。

次の図は、IdP によるアイデンティティプール認証の一般的なサインインセッションを示しています。



アイデンティティプールの認証フロー

1. ユーザーがアプリケーションにアクセスします。
2. [サインイン] リンクを選択します。
3. アプリケーションは、IdP を使用してサインインプロンプトをユーザーに指示します。
4. ユーザー名とパスワードを入力します。
5. IdP はユーザーの認証情報を検証します。
6. IdP は、SAML レスポンスまたは認証コードを使用してユーザーをアプリケーションにリダイレクトします。
7. ユーザーが認証コードを渡した場合、アプリケーションはコードを IdP トークンと交換します。
8. アプリケーションは、ユーザーの JWT またはアサーションについてデコード、検証、保存、キャッシングを行います。
9. アプリケーションは [GetId](#) API リクエストを行うメソッドを呼び出します。ユーザーのトークンまたはアサーションを渡し、アイデンティティ ID をリクエストします。
10. アイデンティティプールは、設定された ID プロバイダーに対してトークンまたはアサーションを検証します。
11. アイデンティティプールはアイデンティティ ID を返します。
12. アプリケーションは、[GetCredentialsForIdentity](#) API リクエストを行うメソッドを呼び出します。ユーザーのトークンまたはアサーションを渡し、IAM ロールをリクエストします。
13. アイデンティティプールは新しい JWT を生成します。新しい JWT には、IAM ロールをリクエストするクレームが含まれています。アイデンティティプールは、ユーザーのリクエストと、IdP のアイデンティティプール設定のロール選択基準に基づいてロールを決定します。
14. AWS Security Token Service (AWS STS) は、ID プールからの [AssumeRoleWithWebIdentity](#) リクエストに応答します。レスポンスには、IAM ロールを持つ一時的セッションの API 認証情報が含まれています。
15. アプリケーションはセッション認証情報を保存します。
16. ユーザーは、AWSでアクセス保護されたリソースを必要とするアクションをアプリケーションで実行します。
17. アプリケーションは、必要な の API リクエスト https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-signing.html に署名として一時的な認証情報を適用します AWS のサービス。
18. IAM は、認証情報のロールにアタッチされたポリシーを評価します。それらをリクエストと比較します。

19. リクエストされたデータ AWS のサービス を返します。

20. アプリケーションは、ユーザーのインターフェイスにデータをレンダリングします。

21. ユーザーはデータを表示します。

バリエーションとカスタマイズ

ユーザープールによる認証を視覚化するには、問題トークン/アサーションステップの後に、以前のユーザープールの概要のいずれかを挿入します。デベロッパー認証は、ID のリクエスト前のすべてのステップを、[デベロッパー認証情報](#)によって署名されたリクエストに置き換えます。また、ゲスト認証は ID のリクエストに直接スキップし、認証を検証せず、[アクセスが制限された IAM ロールの認証情報](#)を返します。

関連リソース

- [Amazon Cognito アイデンティティプール](#)
- [ユーザー IAM ロール](#)
- [ID プールの認証フロー](#)

AWS SDK でのこのサービスの使用

AWS Software Development Kit (SDKs) は、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようになる API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例

SDK ドキュメント	コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback] リンクから、コードの例をリクエストしてください。

Amazon Verified Permissions による承認

[Amazon Verified Permissions](#) は、お客様が構築するアプリケーションの認可サービスです。Amazon Cognito ユーザープールを ID ソースとして追加すると、アプリケーションはユーザープールアクセスまたはアイデンティティ (ID) トークンを Verified Permissions に渡して、許可または拒否を決定できます。Verified Permissions は、[Cedar ポリシー言語](#) で記述したポリシーに基づいてユーザーのプロパティとリクエストコンテキストを考慮します。リクエストコンテキストには、リクエストしたドキュメント、画像、その他のリソースの ID と、ユーザーがリソースに対して実行するアクションを含めることができます。

アプリケーションは、[IsAuthorizedWithToken](#) API リクエストまたは [BatchIsAuthorizedWithToken](#) API リクエストで Verified Permissions にユーザーの ID またはアクセストークンを提供できます。これらの API オペレーションは、ユーザーを Principal として受け入れ、アクセスする

Resource の Action についての認可に関する決定を行います。カスタム Context を追加することで、より詳細なアクセス決定を行うことができます。

アプリケーションが `IsAuthorizedWithToken` API リクエストでトークンを提示すると、Verified Permissions は次の検証を実行します。

1. ユーザープールは、リクエストされたポリシーストアに設定された Verified Permissions の [ID ソース](#)です。
2. アクセストークンまたは ID トークンの `client_id` または `aud` クレームは、それぞれ Verified Permissions に提供したユーザープールアプリケーションのクライアント ID と一致します。このクレームを検証するには、Verified Permissions の ID ソースで[クライアント ID 検証](#)を設定する必要があります。
3. トークンの有効期限は切れていません。
4. トークン内の `token_use` クレームの値は、`IsAuthorizedWithToken` に渡したパラメータと一致します。`token_use` クレームは、`accessToken` パラメータに渡した場合は `access` である必要があり、`identityToken` パラメータに渡した場合は `id` である必要があります。
5. トークンの署名は、ユーザープールの公開された JSON ウェブキー (JWK) から取得されます。JWK は、`https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/jwks.json` で確認できます。

失効したトークンと削除されたユーザー

Verified Permissions は、ID ソースとユーザートークンの有効期限からの既知の情報のみを検証します。Verified Permissions では、トークンの失効やユーザーの存在は確認されません。ユーザーのトークンを取り消したり、ユーザープールからユーザープロファイルを削除したりした場合でも、Verified Permissions は有効期限が切れるまでトークンを有効と見なします。

ポリシーの評価

ユーザープールを[ポリシーストア](#)の [ID ソース](#)として設定します。Verified Permissions へのリクエストでユーザーのトークンを送信するようにアプリケーションを設定します。Verified Permissions はリクエストごとに、トークンのクレームをポリシーと比較します。Verified Permissions は、AWS の IAM ポリシーに似ていて、プリンシパル、リソース、およびアクションを宣言します。Verified Permissions は、リクエストが許可されたアクションと一致し、明示的な Deny アクションと一致しない場合は Allow で応答し、それ以外の場合は Deny で応答します。詳細については、「[Amazon Verified Permissions ユーザーガイド](#)」の「Amazon Verified Permissions のポリシー」を参照してください。

トークンをカスタマイズする

Verified Permissions に提示するユーザークレームを変更、追加、削除するには、アクセストークンと ID トークンの内容を [トークン生成前の Lambda トリガー](#) でカスタマイズします。プレトークンを生成するトリガーを使用すると、トークンでクレームを追加および変更できます。例えば、データベースで追加のユーザー属性をクエリし、それらを ID トークンにエンコードできます。

Note

Verified Permissions がクレームを処理する方法を考慮して、cognito、dev、または custom という名前のクレームをプレトークンの生成機能に追加しないでください。これらの予約済みクレームのプレフィックスを、cognito:username のようにコロンで区切られた形式ではなく、完全なクレーム名として提示すると、承認リクエストは失敗します。

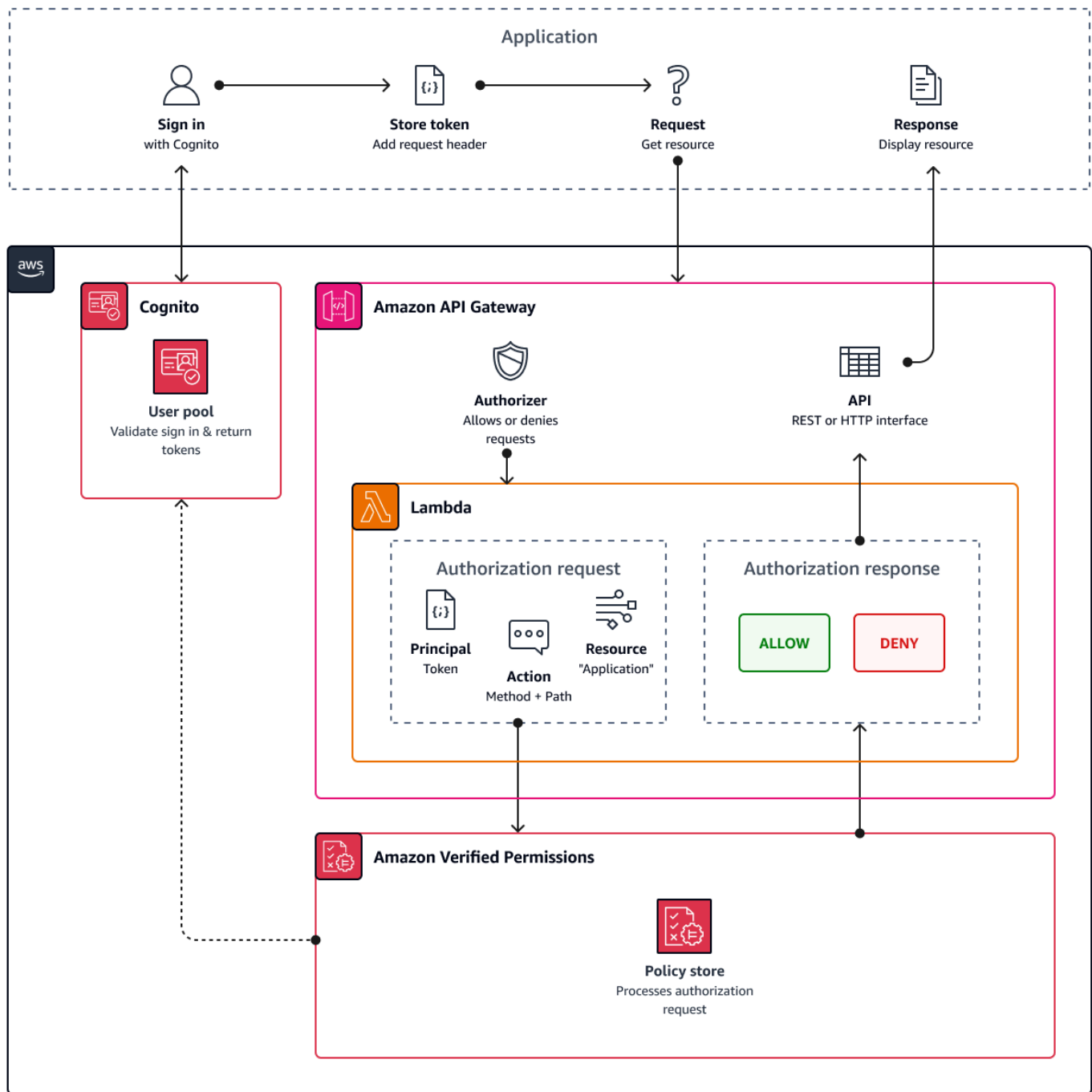
その他のリソース

- [Amazon Cognito トークンを Verified Permissions スキーマにマッピングする](#)
- [Amazon Verified Permissions と Amazon Cognito を使用して API Gateway API を認可する](#)
- [ワークショップ: Amazon Cognito と Verified Permissions による認証と認可](#)

Verified Permissions を使用した API 認可

ID トークンまたはアクセストークンは、Verified Permissions を使用してバックエンド Amazon API Gateway REST API へのリクエストを認可できます。ユーザープールと API への即時リンクを含む [ポリシーストア](#) を作成できます。「[API Gateway と ID ソースを使用してセットアップする](#)」を開始オプションとして使用すると、Verified Permissions はユーザープールの ID ソースをポリシーストアに追加し、Lambda オーソライザーを API に追加します。アプリケーションがユーザープールベアラトークンを API に渡すと、Lambda オーソライザーは Verified Permissions を呼び出します。オーソライザーはトークンをプリンシパルとして、リクエストパスとメソッドをアクションとして渡します。

次の図は、Verified Permissions を使用した API Gateway API 承認フローを示しています。詳細については、「Amazon Verified Permissions ユーザーガイド」の「[API-linked policy stores](#)」を参照してください。



Verified Permissions は、[ユーザープールグループ](#)に関する API 認可を構築します。ID トークンとアクセストークンの両方に `cognito:groups` クレームが含まれているため、ポリシーストアはさまざまなアプリケーションコンテキストで API のロールベースのアクセス制御 (RBAC) を管理できます。

ポリシーストア設定の選択

ポリシーストアで ID ソースを設定するときは、アクセストークンを処理するか、ID トークンを処理するかを選択する必要があります。この決定は、ポリシーエンジンの動作方法にとって重要です。ID トークンにはユーザー属性が含まれます。アクセストークンには、[OAuth スコープ](#)というユーザーアクセスコントロール情報が含まれています。どちらのトークンタイプにもグループメンバー情報が含まれていますが、通常は Verified Permissions ポリシーストアを使用して RBAC のアクセストークンを使用することをお勧めします。アクセストークンは、認可に関する決定に役立つスコープをグループメンバーシップに追加します。アクセストークンのクレームは、認可リクエストの[コンテキスト](#)になります。

ユーザープールを ID ソースとして設定する場合は、ユーザーエンティティタイプとグループエンティティタイプも設定する必要があります。エンティティタイプは、Verified Permissions ポリシーで参照できるプリンシパル、アクション、リソース識別子です。ポリシーストア内のエンティティはメンバーシップ関係を持つことができ、1 つのエンティティを親エンティティのメンバーにすることができます。メンバーシップを使用すると、プリンシパルグループ、アクショングループ、リソースグループを参照できます。ユーザープールグループの場合、指定するユーザーエンティティタイプはグループエンティティタイプのメンバーである必要があります。[API リンクポリシーストア](#)をセットアップするか、Verified Permissions コンソールでガイド付きセットアップに従うと、ポリシーストアは自動的にこの親メンバー関係を持ちます。

ID トークンは、RBAC を属性ベースのアクセス制御 (ABAC) と組み合わせることができます。[API リンクポリシーストア](#)を作成したら、[ユーザー属性](#)とグループメンバーシップを使用してポリシーを強化できます。ID トークンの属性クレームは、認可リクエストの[プリンシパル属性](#)になります。ポリシーは、プリンシパル属性に基づいて認可に関する決定を行うことができます。

また、指定した許容可能なアプリケーションクライアントのリストに一致する aud クレームまたは client_id クレームを持つトークンを受け入れるようにポリシーストアを設定することもできます。

ロールベースの API 認可用ポリシーの例

次のポリシー例は、[PetStore](#) サンプル REST API 用の Verified Permissions ポリシーストアのセットアップによって作成されました。

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

```
);
```

Verified Permissions は、次の場合にアプリケーションからの認可リクエストに Allow 決定を返します。

1. アプリケーションは、Authorization ヘッダー内の ID トークンまたはアクセストークンをベアラトークンとして渡しました。
2. アプリケーションは、文字列 MyGroup を含む cognito:groups クレームでトークンを渡しました。
3. アプリケーションは、https://myapi.example.com/pets または https://myapi.example.com/pets/scrappy などに HTTP GET リクエストを行いました。

Amazon Cognito ユーザーのポリシーの例。

ユーザープールは、API リクエスト以外の条件で Verified Permissions への認可リクエストを生成することもできます。アプリケーション内のアクセスコントロールの決定は、ポリシーストアに送信できます。例えば、リクエストがネットワークを通過する前に、属性ベースのアクセス制御で Amazon DynamoDB または Amazon S3 セキュリティを補完し、クォータの使用量を減らすことができます。

次の例では、[Cedar ポリシー言語](#)を使用して、1 つのユーザープールアプリケーションクライアントで認証する Finance ユーザーに example_image.png の読み取りと書き込みを許可しています。アプリケーションのユーザーである John は、アプリケーションクライアントから ID トークンを受け取り、それを GET リクエストで認可が必要な URLhttps://example.com/images/example_image.pngに渡します。John の ID トークンには、ユーザープールアプリケーションのクライアント ID 1234567890example の aud クレームが含まれています。また、プレトークンを生成する Lambda 関数によって、John 用に新しいクレーム costCenter (値は Finance1234) が挿入されています。

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

次のリクエスト本文の結果は Allow レスポンスになります。

```
{
  "accesstoken": "[John's ID token]",
  "action": {
    "actionId": "readFile",
    "actionType": "Action"
  },
  "resource": {
    "entityId": "example_image.png",
    "entityType": "Photo"
  }
}
```

Verified Permissions でプリンシパルを指定する場合は、次の形式を使用してください。

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

以下は、ID us-east-1_Example、サブ ID (ユーザー ID) 973db890-092c-49e4-a9d0-912a4c0a20c7 を持つユーザープール内のユーザーのプリンシパルの例です。

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

Verified Permissions でユーザーグループを指定する場合は、次の形式を使用してください。

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

属性ベースのアクセスコントロール

アプリの Verified Permissions による認可、および AWS 認証情報の Amazon Cognito ID プールの[アクセスコントロール機能の属性](#)は、どちらも属性ベースのアクセスコントロール (ABAC) の形式で

す。以下は、Verified Permissions と Amazon Cognito ABAC の機能を比較したものです。ABAC では、システムがエンティティの属性を検査し、定義した条件に基づいて承認の決定を行います。

サービス	プロセス	結果
Amazon Verified Permissions	Returns an ## or ## 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 セッションタグ to your user based on their attributes. IAM policy conditions can check tags ## or ## user access to AWS のサービス.	A tagged session with temporary AWS credentials for an IAM role.

AWS SDK を使用した Amazon Cognito のコード例。

次のコード例は、Amazon Cognito を AWS Software Development Kit (SDK) と使用方法を示しています。

AWS SDK デベロッパーガイドとコード例の詳細なリストについては、「[AWS SDK でのこのサービスの使用](#)」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードの例

- [SDK を使用した Amazon Cognito ID のコード例 AWS SDKs](#)
 - [SDK を使用した Amazon Cognito ID の基本的な例 AWS SDKs](#)
 - [SDK を使用した Amazon Cognito ID のアクション AWS SDKs](#)
 - [AWS SDK または CLI CreateIdentityPoolで を使用する](#)
 - [AWS SDK または CLI DeleteIdentityPoolで を使用する](#)
 - [CLI で DescribeIdentityPool を使用する](#)
 - [AWS SDK GetCredentialsForIdentityで を使用する](#)
 - [CLI で GetIdentityPoolRoles を使用する](#)
 - [AWS SDK または CLI ListIdentityPoolsで を使用する](#)
 - [CLI で SetIdentityPoolRoles を使用する](#)
 - [CLI で UpdateIdentityPool を使用する](#)
 - [SDK を使用した Amazon Cognito ID のシナリオ AWS SDKs](#)
 - [Amazon Textract エクスプローラーアプリケーションを作成する](#)
- [SDK を使用した Amazon Cognito ID プロバイダーのコード例 AWS SDKs](#)
 - [AWS SDKs を使用した Amazon Cognito ID プロバイダーの基本的な例](#)
 - [Hello Amazon Cognito](#)
 - [SDK を使用した Amazon Cognito ID プロバイダーのアクション AWS SDKs](#)
 - [AWS SDK または CLI AdminCreateUserで を使用する](#)
 - [AWS SDK または CLI AdminGetUserで を使用する](#)
 - [AWS SDK または CLI AdminInitiateAuthで を使用する](#)
 - [AWS SDK または CLI AdminRespondToAuthChallengeで を使用する](#)
 - [AWS SDK または CLI AdminSetUserPasswordで を使用する](#)

- [AWS SDK または CLI AssociateSoftwareTokenで を使用する](#)
- [AWS SDK または CLI ConfirmDeviceで を使用する](#)
- [AWS SDK または CLI ConfirmForgotPasswordで を使用する](#)
- [AWS SDK または CLI ConfirmSignUpで を使用する](#)
- [AWS SDK または CLI CreateUserPoolで を使用する](#)
- [AWS SDK または CLI CreateUserPoolClientで を使用する](#)
- [AWS SDK または CLI DeleteUserで を使用する](#)
- [AWS SDK または CLI ForgotPasswordで を使用する](#)
- [AWS SDK または CLI InitiateAuthで を使用する](#)
- [AWS SDK または CLI ListUserPoolsで を使用する](#)
- [AWS SDK または CLI ListUsersで を使用する](#)
- [AWS SDK または CLI ResendConfirmationCodeで を使用する](#)
- [AWS SDK または CLI RespondToAuthChallengeで を使用する](#)
- [AWS SDK または CLI SignUpで を使用する](#)
- [AWS SDK または CLI UpdateUserPoolで を使用する](#)
- [AWS SDK または CLI VerifySoftwareTokenで を使用する](#)
- [SDK を使用する Amazon Cognito ID プロバイダーのシナリオ AWS SDKs](#)
 - [AWS SDK を使用して Lambda 関数で既知の Amazon Cognito ユーザーを自動的に確認する](#)
 - [AWS SDK を使用して Lambda 関数を使用して既知の Amazon Cognito ユーザーを自動的に移行する](#)
 - [AWS SDK を使用して MFA を必要とする Amazon Cognito ユーザープールでユーザーをサインアップする](#)
 - [Amazon Cognito のアイデンティティプールと認証フローを使用する](#)
 - [AWS SDK を使用して Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)
- [SDK を使用した Amazon Cognito Sync のコード例 AWS SDKs](#)
 - [SDK を使用した Amazon Cognito Sync の基本的な例 AWS SDKs](#)
 - [SDK を使用した Amazon Cognito Sync のアクション AWS SDKs](#)
 - [AWS SDK ListIdentityPoolUsageで を使用する](#)

SDK を使用した Amazon Cognito ID のコード例 AWS SDKs

次のコード例は、AWS Software Development Kit (SDK) で Amazon Cognito Identity を使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Cognito ID

- [SDK を使用した Amazon Cognito ID の基本的な例 AWS SDKs](#)
 - [SDK を使用した Amazon Cognito ID のアクション AWS SDKs](#)
 - [AWS SDK または CLI CreateIdentityPoolで 使用する](#)
 - [AWS SDK または CLI DeleteIdentityPoolで 使用する](#)
 - [CLI で DescribeIdentityPool を使用する](#)
 - [AWS SDK GetCredentialsForIdentityで 使用する](#)
 - [CLI で GetIdentityPoolRoles を使用する](#)
 - [AWS SDK または CLI ListIdentityPoolsで 使用する](#)
 - [CLI で SetIdentityPoolRoles を使用する](#)
 - [CLI で UpdateIdentityPool を使用する](#)
 - [SDK を使用した Amazon Cognito ID のシナリオ AWS SDKs](#)
 - [Amazon Textract エクスプローラーアプリケーションを作成する](#)

SDK を使用した Amazon Cognito ID の基本的な例 AWS SDKs

次のコード例では、Amazon Cognito ID を AWS SDK と使用する基本的な方法を示しています。

例

- [SDK を使用した Amazon Cognito ID のアクション AWS SDKs](#)
 - [AWS SDK または CLI CreateIdentityPoolで を使用する](#)
 - [AWS SDK または CLI DeleteIdentityPoolで を使用する](#)
 - [CLI で DescribeIdentityPool を使用する](#)
 - [AWS SDK GetCredentialsForIdentityで を使用する](#)
 - [CLI で GetIdentityPoolRoles を使用する](#)
 - [AWS SDK または CLI ListIdentityPoolsで を使用する](#)
 - [CLI で SetIdentityPoolRoles を使用する](#)
 - [CLI で UpdateIdentityPool を使用する](#)

SDK を使用した Amazon Cognito ID のアクション AWS SDKs

次のコード例は、AWS SDKs を使用して個々の Amazon Cognito ID アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

これらは Amazon Cognito Identity API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。アクションは [SDK を使用した Amazon Cognito ID のシナリオ AWS SDKs](#) のコンテキスト内で確認できます。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、[Amazon Cognito ID API Reference](#) (Amazon Cognito ID API リファレンス) を参照してください。

例

- [AWS SDK または CLI CreateIdentityPoolで を使用する](#)
- [AWS SDK または CLI DeleteIdentityPoolで を使用する](#)
- [CLI で DescribeIdentityPool を使用する](#)
- [AWS SDK GetCredentialsForIdentityで を使用する](#)
- [CLI で GetIdentityPoolRoles を使用する](#)
- [AWS SDK または CLI ListIdentityPoolsで を使用する](#)
- [CLI で SetIdentityPoolRoles を使用する](#)
- [CLI で UpdateIdentityPool を使用する](#)

AWS SDK または CLI `CreateIdentityPool`で を使用する

次のサンプルコードは、`CreateIdentityPool` を使用する方法を説明しています。

CLI

AWS CLI

Cognito アイデンティティプールプロバイダーを含むアイデンティティプールを作成するには

この例では `MyIdentityPool` という名前のアイデンティティプールを作成します。これには Cognito アイデンティティプールプロバイダーが含まれます。認証されていないアイデンティティは許可されません。

コマンド:

```
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_aaaaaaaa",ClientId="3n4b5urk1ft4fl3mg5e62d9ado",ServerSideTokenCheck=false
```

出力:

```
{
  "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
    }
  ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[CreateIdentityPool](#)」を参照してください。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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.CognitoIdentityProviderException;

/**
 * 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 "";
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[CreateIdentityPool](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: 認証されていない ID を許可する新しいアイデンティティプールを作成します。

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

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[CreateIdentityPool](#)」を参照してください。

Tools for PowerShell V5

例 1: 認証されていない ID を許可する新しいアイデンティティプールを作成します。

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

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V5)」の「[CreateIdentityPool](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
    }
}
```

- 詳細については、「[AWS SDK for Swift デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for Swift API リファレンス」の「[CreateIdentityPool](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteIdentityPool**で を使用する

次のサンプルコードは、DeleteIdentityPool を使用する方法を説明しています。

CLI

AWS CLI

アイデンティティプールを削除するには

次の delete-identity-pool 例では、指定したアイデンティティプールを削除します。

コマンド:

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

このコマンドでは何も出力されません。

- API の詳細については、AWS CLI コマンドリファレンスの「[DeleteIdentityPool](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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 identityPoolId = args[0];
        CognitoIdentityClient cognitoIdClient = CognitoIdentityClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        deleteIdPool(cognitoIdClient, identityPoolId);
        cognitoIdClient.close();
    }
}
```



```
public static void deleteIdPool(CognitoIdentityClient cognitoIdClient, String
identityPoolId) {
    try {

        DeleteIdentityPoolRequest identityPoolRequest =
DeleteIdentityPoolRequest.builder()
        .identityPoolId(identityPoolId)
        .build();

        cognitoIdClient.deleteIdentityPool(identityPoolRequest);
        System.out.println("Done");

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

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[DeleteIdentityPool](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: 特定のアイデンティティプールを削除します。

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

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[DeleteIdentityPool](#)」を参照してください。

Tools for PowerShell V5

例 1: 特定のアイデンティティプールを削除します。

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

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V5)」の「[DeleteIdentityPool](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import AWSCognitoIdentity

/// 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
    }
}
```

- 詳細については、「[AWS SDK for Swift デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for Swift API リファレンス」の「[DeleteIdentityPool](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

CLI で `DescribeIdentityPool` を使用する

次のサンプルコードは、`DescribeIdentityPool` を使用する方法を説明しています。

CLI

AWS CLI

アイデンティティプールを記述するには

この例では、アイデンティティプールをを記述します。

コマンド:

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

出力:

```
{
  "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
    }
  ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[DescribeIdentityPool](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: 特定のアイデンティティプールに関する情報をその ID を用いて取得します。

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

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[DescribeIdentityPool](#)」を参照してください。

Tools for PowerShell V5

例 1: 特定のアイデンティティプールに関する情報をその ID を用いて取得します。

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

出力:

```
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

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference (V5)」の「[DescribeIdentityPool](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK **GetCredentialsForIdentity**で を使用する

次の例は、GetCredentialsForIdentity を使用する方法を説明しています。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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.CognitoIdentityProviderException;

/**
 * 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);
    }
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[GetCredentialsForIdentity](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

CLI で `GetIdentityPoolRoles` を使用する

次のサンプルコードは、`GetIdentityPoolRoles` を使用する方法を説明しています。

CLI

AWS CLI

アイデンティティプールの役割を取得するには

この例ではアイデンティティプールのロールを取得しています。

コマンド:

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

出力:

```
{
  "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"
  }
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[GetIdentityPoolRoles](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: 特定のアイデンティティプールのロールに関する情報を取得します。

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

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[GetIdentityPoolRoles](#)」を参照してください。

Tools for PowerShell V5

例 1: 特定のアイデンティティプールのロールに関する情報を取得します。

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

出力:

```
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
```


- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference (V5)」の「[GetIdentityPoolRoles](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListIdentityPools**で を使用する

次のサンプルコードは、ListIdentityPools を使用する方法を説明しています。

CLI

AWS CLI

アイデンティティプールを一覧表示するには

この例ではアイデンティティプールを一覧表示します。最大 20 個のアイデンティティが一覧表示されます。

コマンド:

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

出力:

```
{
  "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"
    }
  ]
}
```

```
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListIdentityPools](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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.CognitoIdentityProviderException;

/**
 * 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);
        }
    }
}

```

- APIの詳細については、「AWS SDK for Java 2.x API リファレンス」の「[ListIdentityPools](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: 既存のアイデンティティプールのリストを取得します。

```
Get-CGIIIdentityPoolList
```

出力:

```

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

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[ListIdentityPools](#)」を参照してください。

Tools for PowerShell V5

例 1: 既存のアイデンティティプールのリストを取得します。

```
Get-CGIIIdentityPoolList
```

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V5)」の「[ListIdentityPools](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
    }
}
```

既存のアイデンティティプールの ID を取得します。存在しない場合は、作成します。

```
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
    }
}
```

- 詳細については、「[AWS SDK for Swift デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for Swift API リファレンス」の「[ListIdentityPools](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

CLI で **SetIdentityPoolRoles** を使用する

次のサンプルコードは、SetIdentityPoolRoles を使用する方法を説明しています。

CLI

AWS CLI

アイデンティティプールの役割を設定するには

次の `set-identity-pool-roles` の例では、アイデンティティプールのロールを設定しています。

```
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"
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetIdentityPoolRoles](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: 認証されていない IAM ロールを持つように特定のアイデンティティプールを設定します。

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

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[SetIdentityPoolRoles](#)」を参照してください。

Tools for PowerShell V5

例 1: 認証されていない IAM ロールを持つように特定のアイデンティティプールを設定します。

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

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference (V5)」の「[SetIdentityPoolRoles](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

CLI で `UpdateIdentityPool` を使用する

次のサンプルコードは、`UpdateIdentityPool` を使用する方法を説明しています。

CLI

AWS CLI

アイデンティティプールを更新するには

この例では、アイデンティティプールを更新しています。名前を「MyIdentityPool」に設定します。Cognito をアイデンティティプロバイダとして追加します。認証されていないアイデンティティは許可されません。

コマンド:

```
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="3n4b5urk1ft4fl3mg5e62d9ado",ServerSideTokenCheck=false
```

出力:

```
{
  "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
    }
  ]
}
```


- API の詳細については、「AWS CLI コマンドリファレンス」の「[UpdateIdentityPool](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: アイデンティティプールプロパティの一部を更新します。この場合、アイデンティティプールの名前を更新します。

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

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[UpdateIdentityPool](#)」を参照してください。

Tools for PowerShell V5

例 1: アイデンティティプールプロパティの一部を更新します。この場合、アイデンティティプールの名前を更新します。

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

出力:

```
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
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet Reference (V5)」の「[UpdateIdentityPool](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Cognito ID のシナリオ AWS SDKs

次のコード例は、Amazon Cognito Identity with AWS SDKs で一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Amazon Cognito ID 内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出すことによって特定のタスクを実行する方法を示しています。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- [Amazon Textract エクスプローラーアプリケーションを作成する](#)

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションで Amazon Textract の出力を調べる方法を示しています。

JavaScript

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構

築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

Amazon Textract AWS SDK for Python (Boto3) でを使用して、ドキュメントイメージ内のテキスト、フォーム、テーブル要素を検出する方法を示します。入力イメージと Amazon Textract 出力は、検出された要素を探索できる Tkinter アプリケーションに表示されます。

- Amazon Textract にドキュメントイメージを送信し、検出された要素の出力を調べます。
- Amazon Textract に直接イメージを送信するか、Amazon Simple Storage Service (Amazon S3) バケットを通じてイメージを送信します。
- 非同期 API を使用して、ジョブの完了時に Amazon Simple Notification Service (Amazon SNS) トピックに通知を発行するジョブを開始します。
- Amazon Simple Queue Service (Amazon SQS) キューにジョブ完了メッセージについてポーリングし、結果を表示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3

- Amazon SNS
- Amazon SQS
- Amazon Textract

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Cognito ID プロバイダーのコード例 AWS SDKs

次のコード例は、AWS Software Development Kit (SDK) で Amazon Cognito ID プロバイダーを使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

シナリオは、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Cognito ID プロバイダー

- [AWS SDKs を使用した Amazon Cognito ID プロバイダーの基本的な例](#)
 - [Hello Amazon Cognito](#)
 - [SDK を使用した Amazon Cognito ID プロバイダーのアクション AWS SDKs](#)
 - [AWS SDK または CLI AdminCreateUser で 使用する](#)
 - [AWS SDK または CLI AdminGetUser で 使用する](#)
 - [AWS SDK または CLI AdminInitiateAuth で 使用する](#)
 - [AWS SDK または CLI AdminRespondToAuthChallenge で 使用する](#)
 - [AWS SDK または CLI AdminSetUserPassword で 使用する](#)
 - [AWS SDK または CLI AssociateSoftwareToken で 使用する](#)

- [AWS SDK または CLI ConfirmDevice で 使用する](#)
- [AWS SDK または CLI ConfirmForgotPassword で 使用する](#)
- [AWS SDK または CLI ConfirmSignUp で 使用する](#)
- [AWS SDK または CLI CreateUserPool で 使用する](#)
- [AWS SDK または CLI CreateUserPoolClient で 使用する](#)
- [AWS SDK または CLI DeleteUser で 使用する](#)
- [AWS SDK または CLI ForgotPassword で 使用する](#)
- [AWS SDK または CLI InitiateAuth で 使用する](#)
- [AWS SDK または CLI ListUserPools で 使用する](#)
- [AWS SDK または CLI ListUsers で 使用する](#)
- [AWS SDK または CLI ResendConfirmationCode で 使用する](#)
- [AWS SDK または CLI RespondToAuthChallenge で 使用する](#)
- [AWS SDK または CLI SignUp で 使用する](#)
- [AWS SDK または CLI UpdateUserPool で 使用する](#)
- [AWS SDK または CLI VerifySoftwareToken で 使用する](#)
- [SDK を使用する Amazon Cognito ID プロバイダーのシナリオ AWS SDKs](#)
 - [AWS SDK を使用して Lambda 関数で既知の Amazon Cognito ユーザーを自動的に確認する](#)
 - [AWS SDK を使用して Lambda 関数を使用して既知の Amazon Cognito ユーザーを自動的に移行する](#)
 - [AWS SDK を使用して MFA を必要とする Amazon Cognito ユーザープールでユーザーをサインアップする](#)
 - [Amazon Cognito のアイデンティティプールと認証フローを使用する](#)
 - [AWS SDK を使用して Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

AWS SDKs を使用した Amazon Cognito ID プロバイダーの基本的な例

次のコード例では、Amazon Cognito ID プロバイダーを AWS SDK と使用する基本的な方法を示しています。

例

- [Hello Amazon Cognito](#)

- [SDK を使用した Amazon Cognito ID プロバイダーのアクション AWS SDKs](#)
 - [AWS SDK または CLI AdminCreateUser で 使用する](#)
 - [AWS SDK または CLI AdminGetUser で 使用する](#)
 - [AWS SDK または CLI AdminInitiateAuth で 使用する](#)
 - [AWS SDK または CLI AdminRespondToAuthChallenge で 使用する](#)
 - [AWS SDK または CLI AdminSetUserPassword で 使用する](#)
 - [AWS SDK または CLI AssociateSoftwareToken で 使用する](#)
 - [AWS SDK または CLI ConfirmDevice で 使用する](#)
 - [AWS SDK または CLI ConfirmForgotPassword で 使用する](#)
 - [AWS SDK または CLI ConfirmSignUp で 使用する](#)
 - [AWS SDK または CLI CreateUserPool で 使用する](#)
 - [AWS SDK または CLI CreateUserPoolClient で 使用する](#)
 - [AWS SDK または CLI DeleteUser で 使用する](#)
 - [AWS SDK または CLI ForgotPassword で 使用する](#)
 - [AWS SDK または CLI InitiateAuth で 使用する](#)
 - [AWS SDK または CLI ListUserPools で 使用する](#)
 - [AWS SDK または CLI ListUsers で 使用する](#)
 - [AWS SDK または CLI ResendConfirmationCode で 使用する](#)
 - [AWS SDK または CLI RespondToAuthChallenge で 使用する](#)
 - [AWS SDK または CLI SignUp で 使用する](#)
 - [AWS SDK または CLI UpdateUserPool で 使用する](#)
 - [AWS SDK または CLI VerifySoftwareToken で 使用する](#)

Hello Amazon Cognito

次のコード例は、Amazon Cognito の使用を開始する方法を示しています。

C++

SDK for C++

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# 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})

```

hello_cognito.cpp ソースファイルのコード。

```

#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;
```

```
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[ListUserPools](#)」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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\t%v: %v\n", *pool.Name, *pool.Id)
    }
}
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[ListUserPools](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
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);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ListUserPools](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[ListUserPools](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import boto3

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

# 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.")
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[ListUserPools](#)」を参照してください。

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Ruby API リファレンスの「[ListUserPools](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Cognito ID プロバイダーのアクション AWS SDKs

次のコード例は、AWS SDKs を使用して個々の Amazon Cognito ID プロバイダーアクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

これらは Amazon Cognito Identity Provider API を呼び出すもので、コンテキスト内で実行する必要があります。大規模なプログラムからのコード抜粋です。アクションは [SDK を使用する Amazon Cognito ID プロバイダーのシナリオ AWS SDKs](#) のコンテキスト内で確認できます。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、[Amazon Cognito Identity Provider API Reference](#) (Amazon Cognito Identity Provider API リファレンス) を参照してください。

例

- [AWS SDK または CLI AdminCreateUserで を使用する](#)

- [AWS SDK または CLI AdminGetUser で 使用する](#)
- [AWS SDK または CLI AdminInitiateAuth で 使用する](#)
- [AWS SDK または CLI AdminRespondToAuthChallenge で 使用する](#)
- [AWS SDK または CLI AdminSetUserPassword で 使用する](#)
- [AWS SDK または CLI AssociateSoftwareToken で 使用する](#)
- [AWS SDK または CLI ConfirmDevice で 使用する](#)
- [AWS SDK または CLI ConfirmForgotPassword で 使用する](#)
- [AWS SDK または CLI ConfirmSignUp で 使用する](#)
- [AWS SDK または CLI CreateUserPool で 使用する](#)
- [AWS SDK または CLI CreateUserPoolClient で 使用する](#)
- [AWS SDK または CLI DeleteUser で 使用する](#)
- [AWS SDK または CLI ForgotPassword で 使用する](#)
- [AWS SDK または CLI InitiateAuth で 使用する](#)
- [AWS SDK または CLI ListUserPools で 使用する](#)
- [AWS SDK または CLI ListUsers で 使用する](#)
- [AWS SDK または CLI ResendConfirmationCode で 使用する](#)
- [AWS SDK または CLI RespondToAuthChallenge で 使用する](#)
- [AWS SDK または CLI SignUp で 使用する](#)
- [AWS SDK または CLI UpdateUserPool で 使用する](#)
- [AWS SDK または CLI VerifySoftwareToken で 使用する](#)

AWS SDK または CLI **AdminCreateUser** で 使用する

次のサンプルコードは、AdminCreateUser を 使用する 方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

CLI

AWS CLI

ユーザーを作成するには

次の `admin-create-user` の例では、指定された設定の E メールアドレスと電話番号のユーザーを作成しています。

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

出力:

```
{  
  "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"  
  }  
}
```

- APIの詳細については、「AWS CLI コマンドリファレンス」の「[AdminCreateUser](#)」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「[AdminCreateUser](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **AdminGetUser**で を使用する

次のサンプルコードは、AdminGetUser を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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)
{
    AdminGetUserRequest userRequest = new AdminGetUserRequest
    {
        Username = userName,
        UserPoolId = poolId,
    };

    var response = await _cognitoService.AdminGetUserAsync(userRequest);

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

- API の詳細については、AWS SDK for .NET API リファレンスの「[AdminGetUser](#)」を参照してください。

C++

SDK for C++

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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::AdminGetUserRequest request;
request.SetUsername(userName);
request.SetUserPoolId(userPoolID);

Aws::CognitoIdentityProvider::Model::AdminGetUserOutcome outcome =
    client.AdminGetUser(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::AdminGetUser. "
        << outcome.GetError().GetMessage()
        << std::endl;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[AdminGetUser](#)」を参照してください。

CLI

AWS CLI

ユーザーを取得するには

この例では、ユーザー名 `jane@example.com` に関する情報を取得します。

コマンド:

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

出力:

```
{  
  "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  
}
```

```
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[AdminGetUser](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

        AdminGetUserResponse response =
identityProviderClient.adminGetUser(userRequest);
        System.out.println("User status " + response.userStatusAsString());

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

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[AdminGetUser](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

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

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

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[AdminGetUser](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun getAdminUser(
  userNameVal: String?,
  poolIdVal: String?,
) {
  val userRequest =
```

```

        AdminGetUserRequest {
            username = userNameVal
            userPoolId = poolIdVal
        }

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

```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[AdminGetUser](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

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

```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[AdminGetUser](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

import AWSClientRuntime
import AWSCognitoIdentityProvider

/// Get information about a specific user in a user pool.
///
/// - Parameters:
///   - cipClient: The Amazon Cognito Identity Provider client to use.
///   - userName: The user to retrieve information about.
///   - userPoolId: The user pool to search for the specified user.
///
/// - Returns: `true` if the user's information was successfully
///   retrieved. Otherwise returns `false`.
func adminGetUser(cipClient: CognitoIdentityProviderClient, userName: String,
                  userPoolId: String) async -> Bool {
    do {
        let output = try await cipClient.adminGetUser(
            input: AdminGetUserInput(

```

```
        userPoolId: userPoolId,
        username: userName
    )
)

guard let userStatus = output.userStatus else {
    print("*** Unable to get the user's status.")
    return false
}

print("User status: \(userStatus)")
return true
} catch {
    return false
}
}
```

- APIの詳細については、「AWS SDK for Swift API リファレンス」の「[AdminGetUser](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **AdminInitiateAuth**で を使用する

次のサンプルコードは、AdminInitiateAuth を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[AdminInitiateAuth](#)」を参照してください。

C++

SDK for C++

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[AdminInitiateAuth](#)」を参照してください。

CLI

AWS CLI

管理者としてユーザーのサインインを許可するには

次の `admin-initiate-auth` の例では、ユーザー `diego@example.com` のサインインを許可します。この例には、脅威保護のためのメタデータと Lambda トリガーのための `ClientMetadata` も含まれています。ユーザーは TOTP MFA 用に設定されており、認証を完了する前に、認証アプリケーションからコードの入力を求めるチャレンジを受け取ります。

```
aws cognito-idp admin-initiate-auth \
  --user-pool-id us-west-2_EXAMPLE \
  --client-id 1example23456789 \
  --auth-flow ADMIN_USER_PASSWORD_AUTH \
  --auth-parameters USERNAME=diego@example.com,PASSWORD="My@Example
$Password3!",SECRET_HASH=ExampleEncodedClientIdSecretAndUsername= \
  --context-data="{\"EncodedData\": \"abc123example\", \"HttpHeaders\":
[{\\"headerName\": \"UserAgent\", \"headerValue\": \"Mozilla/5.0 (Windows NT
6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0\"}], \"IpAddress\":
\"192.0.2.1\", \"ServerName\": \"example.com\", \"ServerPath\": \"/login\"}" \
  --client-metadata="{\"MyExampleKey\": \"MyExampleValue\"}"
```

出力:

```
{
  "ChallengeName": "SOFTWARE_TOKEN_MFA",
  "Session": "AYABeExample...",
  "ChallengeParameters": {
    "FRIENDLY_DEVICE_NAME": "MyAuthenticatorApp",
    "USER_ID_FOR_SRP": "diego@example.com"
  }
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[管理認証フロー](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[AdminInitiateAuth](#)」を参照してください。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[AdminInitiateAuth](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[AdminInitiateAuth](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[AdminInitiateAuth](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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

```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[AdminInitiateAuth](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
TRY.
    " Set up authentication parameters
    DATA(lt_auth_params) = VALUE /aws1/
cl_cgpaauthparamstype_w=>tt_authparameterstype(
    ( VALUE /aws1/cl_cgpaauthparamstype_w=>ts_authparameterstype_maprow(
        key = 'USERNAME'
        value = NEW /aws1/cl_cgpaauthparamstype_w( iv_user_name ) ) )
    ( VALUE /aws1/cl_cgpaauthparamstype_w=>ts_authparameterstype_maprow(
        key = 'PASSWORD'
        value = NEW /aws1/cl_cgpaauthparamstype_w( iv_password ) ) )
    ).

    " Add SECRET_HASH if provided
    IF iv_secret_hash IS NOT INITIAL.
        INSERT VALUE #(
            key = 'SECRET_HASH'
            value = NEW /aws1/cl_cgpaauthparamstype_w( iv_secret_hash )
        ) INTO TABLE lt_auth_params.
    ENDIF.

    oo_result = lo_cgp->admininitiateauth(
        iv_userpoolid = iv_user_pool_id
        iv_clientid = iv_client_id
        iv_authflow = 'ADMIN_USER_PASSWORD_AUTH'
        it_authparameters = lt_auth_params
    ).

    DATA(lv_challenge) = oo_result->get_challengename( ).

    IF lv_challenge IS INITIAL.
        MESSAGE 'User successfully signed in.' TYPE 'I'.
    ELSE.
```

```

        MESSAGE |Authentication challenge required: { lv_challenge }.| TYPE
'I'.
    ENDIF.

    CATCH /aws1/cx_cgpusernotfoundex INTO DATA(lo_user_ex).
        MESSAGE |User { iv_user_name } not found.| TYPE 'E'.

    CATCH /aws1/cx_cgpnnotauthorizedex INTO DATA(lo_auth_ex).
        MESSAGE 'Not authorized. Check credentials.' TYPE 'E'.
    ENDTRY.

```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[AdminInitiateAuth](#)を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

import AWSClientRuntime
import AWSCognitoIdentityProvider

/// Begin an authentication session.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - clientId: The app client ID to use.
///   - userName: The username to check.
///   - password: The user's password.
///   - userPoolId: The user pool to use.
///
/// - Returns: The session token associated with this authentication
///   session.
func initiateAuth(cipClient: CognitoIdentityProviderClient, clientId: String,
                  userName: String, password: String,

```

```

        userPoolId: String) async -> String? {
    var authParams: [String: String] = [:]

    authParams["USERNAME"] = userName
    authParams["PASSWORD"] = password

    do {
        let output = try await cipClient.adminInitiateAuth(
            input: AdminInitiateAuthInput(
                authFlow:
CognitoIdentityProviderClientTypes.AuthFlowType.adminUserPasswordAuth,
                authParameters: authParams,
                clientId: clientId,
                userPoolId: userPoolId
            )
        )

        guard let challengeName = output.challengeName else {
            print("*** Invalid response from the auth service.")
            return nil
        }

        print("=====> Response challenge is \(challengeName)")

        return output.session
    } catch _ as UserNotFoundException {
        print("*** The specified username, \(userName), doesn't exist.")
        return nil
    } catch _ as UserNotConfirmedException {
        print("*** The user \(userName) has not been confirmed.")
        return nil
    } catch {
        print("*** An unexpected error occurred.")
        return nil
    }
}

```

- APIの詳細については、「AWS SDK for Swift API リファレンス」の「[AdminInitiateAuth](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI `AdminRespondToAuthChallenge`で を使用する

次のサンプルコードは、`AdminRespondToAuthChallenge` を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[AdminRespondToAuthChallenge](#)」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
    }
}
```

- APIの詳細については、「AWS SDK for C++ API リファレンス」の「[AdminRespondToAuthChallenge](#)」を参照してください。

CLI

AWS CLI

認証チャレンジに応答するには

認証チャレンジに応答する方法は、認証フロー、ユーザープールの設定、ユーザーの設定によって異なります。次の `admin-respond-to-auth-challenge` の例では、`diego@example.com` の TOTP MFA コードを提供し、サインインを完了する方法を示します。このユーザープールではデバイスの記憶がオンになっているため、認証結果として新しいデバイスキーも返されます。

```
aws cognito-idp admin-respond-to-auth-challenge \
  --user-pool-id us-west-2_EXAMPLE \
  --client-id 1example23456789 \
  --challenge-name SOFTWARE_TOKEN_MFA \
  --challenge-
responses USERNAME=diego@example.com,SOFTWARE_TOKEN_MFA_CODE=000000 \
  --session AYABeExample...
```

出力:

```
{
  "ChallengeParameters": {},
  "AuthenticationResult": {
    "AccessToken": "eyJra456defEXAMPLE",
    "ExpiresIn": 3600,
    "TokenType": "Bearer",
    "RefreshToken": "eyJra123abcEXAMPLE",
    "IdToken": "eyJra789ghiEXAMPLE",
    "NewDeviceMetadata": {
      "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
      "DeviceGroupKey": "-ExAmPlE1"
    }
  }
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[管理認証フロー](#)」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「[AdminRespondToAuthChallenge](#)」を参照してください。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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());
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[AdminRespondToAuthChallenge](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[AdminRespondToAuthChallenge](#)」を参照してください。

Kotlin

SDK for Kotlin

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

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

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[AdminRespondToAuthChallenge](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

MFA のチャレンジに応答するには、関連する MFA アプリケーションによって生成されたコードを入手します。

```
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

```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[AdminRespondToAuthChallenge](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
TRY.
    " Build challenge responses
    DATA(lt_challenge_responses) = VALUE /aws1/
cl_cgpchallengerspstyp00=>tt_challengerresponsestype(
    ( VALUE /aws1/cl_cgpchallengerspstyp00=>ts_challengerspstype_maprow(
        key = 'USERNAME'
        value = NEW /aws1/cl_cgpchallengerspstyp00( iv_user_name ) ) )
    ( VALUE /aws1/cl_cgpchallengerspstyp00=>ts_challengerspstype_maprow(
        key = 'SOFTWARE_TOKEN_MFA_CODE'
        value = NEW /aws1/cl_cgpchallengerspstyp00( iv_mfa_code ) ) )
    ).

    " Add SECRET_HASH if provided
    IF iv_secret_hash IS NOT INITIAL.
        INSERT VALUE #(
            key = 'SECRET_HASH'
            value = NEW /aws1/cl_cgpchallengerspstyp00( iv_secret_hash )
        ) INTO TABLE lt_challenge_responses.
    ENDIF.

    DATA(lo_result) = lo_cgp->adminrespondtoauthchallenge(
        iv_userpoolid = iv_user_pool_id
        iv_clientid = iv_client_id
        iv_challenge_name = 'SOFTWARE_TOKEN_MFA'
        it_challengeresponses = lt_challenge_responses
```

```

        iv_session = iv_session
    ).

    oo_auth_result = lo_result->get_authenticationresult( ).

    IF oo_auth_result IS BOUND.
        MESSAGE 'MFA challenge completed successfully.' TYPE 'I'.
    ELSE.
        " Another challenge might be required
        DATA(lv_next_challenge) = lo_result->get_challengename( ).
        MESSAGE |Additional challenge required: { lv_next_challenge }.| TYPE
'I'.
    ENDIF.

    CATCH /aws1/cx_cgpcodemismatchex INTO DATA(lo_code_ex).
        MESSAGE 'Invalid MFA code provided.' TYPE 'E'.

    CATCH /aws1/cx_cgpxpiredcodeex INTO DATA(lo_expired_ex).
        MESSAGE 'MFA code has expired.' TYPE 'E'.

    CATCH /aws1/cx_cgpnnotauthorizedex INTO DATA(lo_auth_ex).
        MESSAGE 'Not authorized. Check MFA configuration.' TYPE 'E'.
    ENDTRY.

```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの [AdminRespondToAuthChallenge](#) を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

import AWSClientRuntime
import AWSCognitoIdentityProvider

```

```

/// Respond to the authentication challenge received from Cognito after
/// initiating an authentication session. This involves sending a current
/// MFA code to the service.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - userName: The user's username.
///   - clientId: The app client ID.
///   - userPoolId: The user pool to sign into.
///   - mfaCode: The 6-digit MFA code currently displayed by the user's
///     authenticator.
///   - session: The authentication session to continue processing.
func adminRespondToAuthChallenge(cipClient: CognitoIdentityProviderClient,
    userName: String,
                                clientId: String, userPoolId: String,
mfaCode: String,
                                session: String) async {
    print("=====> SOFTWARE_TOKEN_MFA challenge is generated...")

    var challengeResponsesOb: [String: String] = [:]
    challengeResponsesOb["USERNAME"] = userName
    challengeResponsesOb["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

    do {
        let output = try await cipClient.adminRespondToAuthChallenge(
            input: AdminRespondToAuthChallengeInput(
                challengeName:
CognitoIdentityProviderClientTypes.ChallengeNameType.softwareTokenMfa,
                challengeResponses: challengeResponsesOb,
                clientId: clientId,
                session: session,
                userPoolId: userPoolId
            )
        )

        guard let authenticationResult = output.authenticationResult else {
            print("*** Unable to get authentication result.")
            return
        }

        print("=====> Authentication result (JWTs are redacted):")
        print(authenticationResult)
    } catch _ as SoftwareTokenMFANotFoundException {
        print("*** The specified user pool isn't configured for MFA.")
    }
}

```

```

        return
    } catch _ as CodeMismatchException {
        print("*** The specified MFA code doesn't match the expected value.")
        return
    } catch _ as UserNotFoundException {
        print("*** The specified username, \(userName), doesn't exist.")
        return
    } catch _ as UserNotConfirmedException {
        print("*** The user \(userName) has not been confirmed.")
        return
    } catch let error as NotAuthorizedException {
        print("*** Unauthorized access. Reason: \(error.properties.message ??
"<unknown>")")
    } catch {
        print("*** Error responding to the MFA challenge.")
        return
    }
}

```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[AdminRespondToAuthChallenge](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **AdminSetUserPassword**で を使用する

次のサンプルコードは、AdminSetUserPassword を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

CLI

AWS CLI

管理者としてユーザーのパスワードを設定するには

次の `admin-set-user-password` の例では、`diego@example.com` のパスワードを永続的に設定します。

```
aws cognito-idp admin-set-user-password \  
  --user-pool-id us-west-2_EXAMPLE \  
  --username diego@example.com \  
  --password MyExamplePassword1! \  
  --permanent
```

このコマンドでは何も出力されません。

詳細については、「Amazon Cognito デベロッパーガイド」の「[パスワード、パスワード復旧、パスワードポリシー](#)」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「[AdminSetUserPassword](#)」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「[AdminSetUserPassword](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **AssociateSoftwareToken**で を使用する

次のサンプルコードは、AssociateSoftwareToken を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[AssociateSoftwareToken](#)」を参照してください。

C++

SDK for C++

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
#ifdef 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;
    }
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[AssociateSoftwareToken](#)」を参照してください。

CLI

AWS CLI

MFA Authenticator アプリケーション用のシークレットキーを生成するには

次の `associate-software-token` の例では、サインインしてアクセストークンを受け取ったユーザーの TOTP シークレットキーを生成します。生成されたシークレットキーは、認証アプリケーションに手動で入力することも、ユーザーがスキャン可能な QR コードとしてアプリケーションでレンダリングすることもできます。

```
aws cognito-idp associate-software-token \
    --access-token eyJra456defEXAMPLE
```

出力:

```
{
  "SecretCode": "QWERTYUIOP123456EXAMPLE"
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[TOTP ソフトウェアトークン MFA](#)」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「[AssociateSoftwareToken](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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();
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[AssociateSoftwareToken](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
const associateSoftwareToken = (session) => {
```

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

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

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[AssociateSoftwareToken](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun getSecretForAppMFA(sessionVal: String?): String? {
    val softwareTokenRequest =
        AssociateSoftwareTokenRequest {
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { 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
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[AssociateSoftwareToken](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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

```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[AssociateSoftwareToken](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

TRY.
  DATA(lo_result) = lo_cgp->associatesoftwaretoken(
    iv_session = iv_session
  ).

  ov_secret_code = lo_result->get_secretcode( ).

  MESSAGE 'MFA secret code generated successfully.' TYPE 'I'.

CATCH /aws1/cx_cgppresourcenotfoundex INTO DATA(lo_ex).
  MESSAGE 'Session not found or expired.' TYPE 'E'.

```

```
CATCH /aws1/cx_cgpnnotauthorizedex INTO DATA(lo_auth_ex).  
  MESSAGE 'Not authorized to associate software token.' TYPE 'E'.  
ENDTRY.
```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[AssociateSoftwareToken](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import AWSClientRuntime  
import AWSCognitoIdentityProvider  
  
/// Request and display an MFA secret token that the user should enter  
/// into their authenticator to set it up for the user account.  
///  
/// - Parameters:  
///   - cipClient: The `CognitoIdentityProviderClient` to use.  
///   - authSession: The authentication session to request an MFA secret  
///     for.  
///  
/// - Returns: A string containing the MFA secret token that should be  
///   entered into the authenticator software.  
func getSecretForAppMFA(cipClient: CognitoIdentityProviderClient,  
authSession: String?) async -> String? {  
    do {  
        let output = try await cipClient.associateSoftwareToken(  
            input: AssociateSoftwareTokenInput(  
                session: authSession  
            )  
        )  
    }  
}
```

```
        guard let secretCode = output.secretCode else {
            print("*** Unable to get the secret code")
            return nil
        }

        print("=====> Enter this token into Google Authenticator:
\\(secretCode)")
        return output.session
    } catch _ as SoftwareTokenMFANotFoundException {
        print("*** The specified user pool isn't configured for MFA.")
        return nil
    } catch {
        print("*** An unexpected error occurred getting the secret for the
app's MFA.")
        return nil
    }
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[AssociateSoftwareToken](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ConfirmDevice**で を使用する

次のサンプルコードは、ConfirmDevice を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[ConfirmDevice](#)」を参照してください。

CLI

AWS CLI

ユーザーデバイスを確認するには

次の `confirm-device` の例では、現在のユーザー用に新しく記憶されたデバイスを追加します。

```
aws cognito-idp confirm-device \  
  --access-token eyJra456defEXAMPLE \  
  --device-key us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \  
  --device-secret-verifier-  
config PasswordVerifier=TXlWZXJpZmllc1N0cmLuZw,Salt=TXlTUlBTYWx0
```

出力:

```
{  
  "UserConfirmationNecessary": false  
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[ユーザープール内のユーザーデバイスの使用](#)」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「[ConfirmDevice](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);  
};
```

- APIの詳細については、AWS SDK for JavaScript API リファレンスの「[ConfirmDevice](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[ConfirmDevice](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ConfirmForgotPassword**で を使用する

次のサンプルコードは、ConfirmForgotPassword を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Lambda 関数を使用して登録済みのユーザーを自動的に移行する](#)

CLI

AWS CLI

忘れたパスワードを確認するには

この例では、ユーザー名 `diego@example.com` が忘れたパスワードを確認しています。

コマンド:

```
aws cognito-idp confirm-forgot-password --client-id 3n4b5urk1ft4fl3mg5e62d9ado --username=diego@example.com --password PASSWORD --confirmation-code CONF_CODE
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ConfirmForgotPassword](#)」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「[ConfirmForgotPassword](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ConfirmSignUp**で を使用する

次のサンプルコードは、ConfirmSignUp を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```


- API の詳細については、AWS SDK for .NET API リファレンスの「[ConfirmSignUp](#)」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[ConfirmSignUp](#)」を参照してください。

CLI

AWS CLI

サインアップを確認するには

この例では、ユーザー名 `diego@example.com` のサインアップを確認します。

コマンド:

```
aws cognito-idp confirm-sign-up --client-id 3n4b5urk1ft4fl3mg5e62d9ado --  
username=diego@example.com --confirmation-code CONF_CODE
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ConfirmSignUp](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ConfirmSignUp](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
const confirmSignUp = ({ clientId, username, code }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new ConfirmSignUpCommand({
        ClientId: clientId,
        Username: username,
        ConfirmationCode: code,
    });

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

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[ConfirmSignUp](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun confirmSignUp(
    clientIdVal: String?,
    codeVal: String?,
    userNameVal: String?,
) {
    val signUpRequest =
        ConfirmSignUpRequest {
            clientId = clientIdVal
            confirmationCode = codeVal
            username = userNameVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
    { identityProviderClient ->
        identityProviderClient.confirmSignUp(signUpRequest)
        println("$userNameVal was confirmed")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[ConfirmSignUp](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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

```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[ConfirmSignUp](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

import AWSClientRuntime
import AWSCognitoIdentityProvider

/// Submit a confirmation code for the specified user. This is the code as
/// entered by the user after they've received it by email or text
/// message.
///
/// - Parameters:

```

```
/// - cipClient: The `CognitoIdentityProviderClient` to use.
/// - clientId: The app client ID the user is signing up for.
/// - userName: The username of the user whose code is being sent.
/// - code: The user's confirmation code.
///
/// - Returns: `true` if the code was successfully confirmed; otherwise
`false`.
func confirmSignUp(cipClient: CognitoIdentityProviderClient, clientId:
String,
                  userName: String, code: String) async -> Bool {
    do {
        _ = try await cipClient.confirmSignUp(
            input: ConfirmSignUpInput(
                clientId: clientId,
                confirmationCode: code,
                username: userName
            )
        )

        print("=====> \(userName) has been confirmed.")
        return true
    } catch {
        print("=====> \(userName)'s code was entered incorrectly.")
        return false
    }
}
```

- APIの詳細については、「AWS SDK for Swift API リファレンス」の「[ConfirmSignUp](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateUserPool**で を使用する

次のサンプルコードは、CreateUserPool を使用する方法を説明しています。

CLI

AWS CLI

最小設定のユーザープールを作成するには

この例では、デフォルト値を使用して MyUserPool という名前のユーザープールを作成します。必要な属性やアプリケーションクライアントはありません。MFA およびアドバンスドセキュリティは無効化されています。

コマンド:

```
aws cognito-idp create-user-pool --pool-name MyUserPool
```

出力:

```
{
  "UserPool": {
    "SchemaAttributes": [
      {
        "Name": "sub",
        "StringAttributeConstraints": {
          "MinLength": "1",
          "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": true,
        "AttributeDataType": "String",
        "Mutable": false
      },
      {
        "Name": "name",
        "StringAttributeConstraints": {
          "MinLength": "0",
          "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
      },
      {
        "Name": "given_name",
```



```
    "StringAttributeConstraints": {
      "MinLength": "0",
      "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "String",
    "Mutable": true
  },
  {
    "Name": "family_name",
    "StringAttributeConstraints": {
      "MinLength": "0",
      "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "String",
    "Mutable": true
  },
  {
    "Name": "middle_name",
    "StringAttributeConstraints": {
      "MinLength": "0",
      "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "String",
    "Mutable": true
  },
  {
    "Name": "nickname",
    "StringAttributeConstraints": {
      "MinLength": "0",
      "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "String",
    "Mutable": true
  },
  {
    "Name": "preferred_username",
```

```
    "StringAttributeConstraints": {
      "MinLength": "0",
      "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "String",
    "Mutable": true
  },
  {
    "Name": "profile",
    "StringAttributeConstraints": {
      "MinLength": "0",
      "MaxLength": "2048"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "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
  },
  {
    "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,
        "AttributeDataType": "String",
        "Mutable": true
    }
```

```
    },
    {
      "Name": "updated_at",
      "NumberAttributeConstraints": {
        "MinValue": "0"
      },
      "DeveloperOnlyAttribute": false,
      "Required": false,
      "AttributeDataType": "Number",
      "Mutable": true
    }
  ],
  "MfaConfiguration": "OFF",
  "Name": "MyUserPool",
  "LastModifiedDate": 1547833345.777,
  "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": {}
}
}
```

2つの必須属性でユーザープールを作成するには

この例では、ユーザープール MyUserPool を作成します。プールは、E メールをユーザー名属性として受け入れるように設定されています。また、Amazon Simple Email Service を使用して、Eメールの送信元アドレスを検証済みのアドレスに設定します。

コマンド:

```
aws cognito-idp 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"
```

出力:

```
{
  "UserPool": {
    "SchemaAttributes": [
      {
        "Name": "sub",
        "StringAttributeConstraints": {
          "MinLength": "1",
          "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": true,
        "AttributeDataType": "String",
        "Mutable": false
      },
      {
        "Name": "name",
        "StringAttributeConstraints": {
          "MinLength": "0",
          "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
      },
      {
        "Name": "given_name",
        "StringAttributeConstraints": {
          "MinLength": "0",
          "MaxLength": "2048"
        },
        "DeveloperOnlyAttribute": false,
        "Required": false,
        "AttributeDataType": "String",
        "Mutable": true
      }
    ]
  }
}
```

```
{
  "Name": "family_name",
  "StringAttributeConstraints": {
    "MinLength": "0",
    "MaxLength": "2048"
  },
  "DeveloperOnlyAttribute": false,
  "Required": false,
  "AttributeDataType": "String",
  "Mutable": true
},
{
  "Name": "middle_name",
  "StringAttributeConstraints": {
    "MinLength": "0",
    "MaxLength": "2048"
  },
  "DeveloperOnlyAttribute": false,
  "Required": false,
  "AttributeDataType": "String",
  "Mutable": true
},
{
  "Name": "nickname",
  "StringAttributeConstraints": {
    "MinLength": "0",
    "MaxLength": "2048"
  },
  "DeveloperOnlyAttribute": false,
  "Required": false,
  "AttributeDataType": "String",
  "Mutable": true
},
{
  "Name": "preferred_username",
  "StringAttributeConstraints": {
    "MinLength": "0",
    "MaxLength": "2048"
  },
  "DeveloperOnlyAttribute": false,
  "Required": false,
  "AttributeDataType": "String",
  "Mutable": true
},
}
```

```
{
  "Name": "profile",
  "StringAttributeConstraints": {
    "MinLength": "0",
    "MaxLength": "2048"
  },
  "DeveloperOnlyAttribute": false,
  "Required": false,
  "AttributeDataType": "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
},
{
  "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,
    "AttributeDataType": "String",
    "Mutable": true
},
{
    "Name": "updated_at",
    "NumberAttributeConstraints": {
        "MinValue": "0"
    },
    "DeveloperOnlyAttribute": false,
    "Required": false,
    "AttributeDataType": "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": {}
  }
}
```

- APIの詳細については、AWS CLI コマンドリファレンスの「[CreateUserPool](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
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 = ""

            Usage:
                <userPoolName>\s

            Where:
                userPoolName - The name to give your user pool when it's
created.

        """;
```

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

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

        String id = createPool(cognitoClient, userPoolName);
        System.out.println("User pool ID: " + id);
        cognitoClient.close();
    }

    public static String createPool(CognitoIdentityProviderClient cognitoClient,
        String userPoolName) {
        try {
            CreateUserPoolRequest request = CreateUserPoolRequest.builder()
                .poolName(userPoolName)
                .build();

            CreateUserPoolResponse response =
            cognitoClient.createUserPool(request);
            return response.userPool().id();

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

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[CreateUserPool](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI `CreateUserPoolClient` で 使用する

次のサンプルコードは、`CreateUserPoolClient` を使用する方法を説明しています。

CLI

AWS CLI

ユーザープールのクライアントを作成する

次の `create-user-pool-client` の例では、クライアントシークレット、明示的な読み取りおよび書き込み属性、ユーザー名とパスワードおよび SRP フローでのサインイン、3 つの IdP でのサインイン、OAuth スコープのサブセットへのアクセス、PinPoint 分析、拡張認証セッション有効期間を設定した新しいユーザープールクライアントを作成します。

```
aws cognito-idp create-user-pool-client \
  --user-pool-id us-west-2_EXAMPLE \
  --client-name MyTestClient \
  --generate-secret \
  --refresh-token-validity 10 \
  --access-token-validity 60 \
  --id-token-validity 60 \
  --token-validity-units AccessToken=minutes,IdToken=minutes,RefreshToken=days \
  --read-attributes email phone_number email_verified phone_number_verified \
  --write-attributes email phone_number \
  --explicit-auth-flows ALLOW_USER_PASSWORD_AUTH ALLOW_USER_SRP_AUTH ALLOW_REFRESH_TOKEN_AUTH \
  --supported-identity-providers Google Facebook MyOIDC \
  --callback-urls https://www.amazon.com https://example.com http://localhost:8001 myapp://example \
  --allowed-o-auth-flows code implicit \
  --allowed-o-auth-scopes openid profile aws.cognito.signin.user.admin solar-system-data/asteroids.add \
  --allowed-o-auth-flows-user-pool-client \
  --analytics-configuration ApplicationArn=arn:aws:mobiletargeting:us-west-2:767671399759:apps/thisisanexamplepinpointapplicationid,UserDataShared=TRUE \
  --prevent-user-existence-errors ENABLED \
  --enable-token-revocation \
  --enable-propagate-additional-user-context-data \
  --auth-session-validity 4
```

出力:

```
{
  "UserPoolClient": {
    "UserPoolId": "us-west-2_EXAMPLE",
    "ClientName": "MyTestClient",
    "ClientId": "123abc456defEXAMPLE",
    "ClientSecret": "this1234is5678my91011example1213client1415secret",
    "LastModifiedDate": 1726788459.464,
    "CreationDate": 1726788459.464,
    "RefreshTokenValidity": 10,
    "AccessTokenValidity": 60,
    "IdTokenValidity": 60,
    "TokenValidityUnits": {
      "AccessToken": "minutes",
      "IdToken": "minutes",
      "RefreshToken": "days"
    },
    "ReadAttributes": [
      "email_verified",
      "phone_number_verified",
      "phone_number",
      "email"
    ],
    "WriteAttributes": [
      "phone_number",
      "email"
    ],
    "ExplicitAuthFlows": [
      "ALLOW_USER_PASSWORD_AUTH",
      "ALLOW_USER_SRP_AUTH",
      "ALLOW_REFRESH_TOKEN_AUTH"
    ],
    "SupportedIdentityProviders": [
      "Google",
      "MyOIDC",
      "Facebook"
    ],
    "CallbackURLs": [
      "https://example.com",
      "https://www.amazon.com",
      "myapp://example",
      "http://localhost:8001"
    ],
  },
}
```

```

    "AllowedOAuthFlows": [
      "implicit",
      "code"
    ],
    "AllowedOAuthScopes": [
      "aws.cognito.signin.user.admin",
      "openid",
      "profile",
      "solar-system-data/asteroids.add"
    ],
    "AllowedOAuthFlowsUserPoolClient": true,
    "AnalyticsConfiguration": {
      "ApplicationArn": "arn:aws:mobiletargeting:us-
west-2:123456789012:apps/thisisanexamplepinpointapplicationid",
      "RoleArn": "arn:aws:iam::123456789012:role/aws-service-role/cognito-
idp.amazonaws.com/AWSServiceRoleForAmazonCognitoIdp",
      "UserDataShared": true
    },
    "PreventUserExistenceErrors": "ENABLED",
    "EnableTokenRevocation": true,
    "EnablePropagateAdditionalUserContextData": true,
    "AuthSessionValidity": 4
  }
}

```

詳細については、「Amazon Cognito デベロッパーガイド」の「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[CreateUserPoolClient](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
```



```
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientResponse;

/**
 * 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);
    }
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[CreateUserPoolClient](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteUser**で を使用する

次のサンプルコードは、DeleteUser を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Lambda 関数を使用して登録済みのユーザーを自動的に確認する](#)
- [Lambda 関数を使用して登録済みのユーザーを自動的に移行する](#)
- [Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;  
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[DeleteUser](#)」を参照してください。

CLI

AWS CLI

ユーザーを削除するには

この例では、ユーザーを削除します。

コマンド:

```
aws cognito-idp delete-user --access-token ACCESS_TOKEN
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[DeleteUser](#)」を参照してください。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- APIの詳細については、「AWS SDK for Go API リファレンス」の「[DeleteUser](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/**
 * 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];
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[DeleteUser](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ForgotPassword**で を使用する

次のサンプルコードは、ForgotPassword を使用方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Lambda 関数を使用して登録済みのユーザーを自動的に移行する](#)

CLI

AWS CLI

パスワードの変更を強制するには

次の forgot-password の例では、パスワードを変更するようにメッセージを jane@example.com に送信しています。

```
aws cognito-idp forgot-password --client-id 38fjsnc484p94kpqsnet7mpld0 --  
username jane@example.com
```

出力:

```
{  
  "CodeDeliveryDetails": {  
    "Destination": "j***@e***.com",  
    "DeliveryMedium": "EMAIL",  
    "AttributeName": "email"  
  }  
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ForgotPassword](#)」を参照してください。

Go

SDK for Go V2

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- APIの詳細については、「AWS SDK for Go API リファレンス」の「[ForgotPassword](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **InitiateAuth**で を使用する

次のサンプルコードは、InitiateAuth を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Lambda 関数を使用して登録済みのユーザーを自動的に確認する](#)
- [Lambda 関数を使用して登録済みのユーザーを自動的に移行する](#)
- [MFA を必要とするユーザープールでユーザーをサインアップする](#)
- [Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[InitiateAuth](#)」を参照してください。

CLI

AWS CLI

ユーザーをサインインするには

次の `initiate-auth` の例では、基本的なユーザー名-パスワードフローを使用してユーザーにサインインし、追加のチャレンジはありません。

```
aws cognito-idp initiate-auth \
  --auth-flow USER_PASSWORD_AUTH \
  --client-id 1example23456789 \
  --analytics-metadata AnalyticsEndpointId=d70b2ba36a8c4dc5a04a0451aEXAMPLE \
  --auth-parameters USERNAME=testuser,PASSWORD=[Password] --user-context-
data EncodedData=mycontextdata --client-metadata MyTestKey=MyTestValue
```

出力:

```
{
  "AuthenticationResult": {
    "AccessToken": "eyJra456defEXAMPLE",
    "ExpiresIn": 3600,
    "TokenType": "Bearer",
    "RefreshToken": "eyJra123abcEXAMPLE",
    "IdToken": "eyJra789ghiEXAMPLE",
    "NewDeviceMetadata": {
      "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
      "DeviceGroupKey": "-v7w9UcY6"
    }
  }
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Authentication](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[InitiateAuth](#)」を参照してください。

Go

SDK for Go V2

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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  
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[InitiateAuth](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);  
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[InitiateAuth](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

この例は、トラッキング対象デバイスを使用して認証を開始する方法を示しています。サインインを完了するには、クライアントはセキュアリモートパスワード (SRP) チャレンジに正しく応答する必要があります。

```
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 {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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[InitiateAuth](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListUserPools**で を使用する

次のサンプルコードは、ListUserPools を使用する方法を説明しています。

.NET

SDK for .NET (v4)

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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.Paginators.ListUserPools(new
ListUserPoolsRequest());

    await foreach (var response in userPoolsPaginator.Responses)
    {
        userPools.AddRange(response.UserPools);
    }
}
```



```
    return userPools;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[ListUserPools](#)」を参照してください。

CLI

AWS CLI

ユーザープールを一覧表示するには

次の `list-user-pools` 例では、現在の CLI 認証情報の AWS アカウントで使用可能なユーザープールの 3 つを一覧表示します。

```
aws cognito-idp list-user-pools \
  --max-results 3
```

出力:

```
{
  "NextToken": "[Pagination token]",
  "UserPools": [
    {
      "CreationDate": 1681502497.741,
      "Id": "us-west-2_EXAMPLE1",
      "LambdaConfig": {
        "CustomMessage": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",
        "PreSignUp": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",
        "PreTokenGeneration": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",
        "PreTokenGenerationConfig": {
          "LambdaArn": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",
          "LambdaVersion": "V1_0"
        }
      }
    },
  ],
}
```

```
        "LastModifiedDate": 1681502497.741,
        "Name": "user pool 1"
    },
    {
        "CreationDate": 1686064178.717,
        "Id": "us-west-2_EXAMPLE2",
        "LambdaConfig": {
        },
        "LastModifiedDate": 1686064178.873,
        "Name": "user pool 2"
    },
    {
        "CreationDate": 1627681712.237,
        "Id": "us-west-2_EXAMPLE3",
        "LambdaConfig": {
            "UserMigration": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction"
        },
        "LastModifiedDate": 1678486942.479,
        "Name": "user pool 3"
    }
  ]
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Amazon Cognito user pools](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListUserPools](#)」を参照してください。

Go

SDK for Go V2

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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)
}
}
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[ListUserPools](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
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);
    }
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[ListUserPools](#)」を参照してください。

Rust

SDK for Rust

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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_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(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「[ListUserPools](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListUsers**で を使用する

次のサンプルコードは、ListUsers を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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.Paginators.ListUsers(request);
    await foreach (var response in usersPaginator.Responses)
    {
        users.AddRange(response.Users);
    }

    return users;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[ListUsers](#)」を参照してください。

CLI

AWS CLI

例 1: サーバー側のフィルターを使用してユーザーを一覧表示するには

次の list-users の例では、リクエストされたユーザープールの E メールアドレスが testuser で始まる 3 人のユーザーを一覧表示します。

```
aws cognito-idp list-users \
  --user-pool-id us-west-2_EXAMPLE \
  --filter email^=\"testuser\" \
  --max-items 3
```

出力:

```
{
  "PaginationToken": "efgh5678EXAMPLE",
  "Users": [
    {
      "Attributes": [
        {
          "Name": "sub",
          "Value": "eaad0219-2117-439f-8d46-4db20e59268f"
        },
        {
          "Name": "email",
          "Value": "testuser@example.com"
        }
      ],
      "Enabled": true,
      "UserCreateDate": 1682955829.578,
      "UserLastModifiedDate": 1689030181.63,
      "UserStatus": "CONFIRMED",
      "Username": "testuser"
    },
    {
```



```
    "Attributes": [
      {
        "Name": "sub",
        "Value": "3b994cfd-0b07-4581-be46-3c82f9a70c90"
      },
      {
        "Name": "email",
        "Value": "testuser2@example.com"
      }
    ],
    "Enabled": true,
    "UserCreateDate": 1684427979.201,
    "UserLastModifiedDate": 1684427979.201,
    "UserStatus": "UNCONFIRMED",
    "Username": "testuser2"
  },
  {
    "Attributes": [
      {
        "Name": "sub",
        "Value": "5929e0d1-4c34-42d1-9b79-a5ecacfe66f7"
      },
      {
        "Name": "email",
        "Value": "testuser3@example.com"
      }
    ],
    "Enabled": true,
    "UserCreateDate": 1684427823.641,
    "UserLastModifiedDate": 1684427823.641,
    "UserStatus": "UNCONFIRMED",
    "Username": "testuser3@example.com"
  }
]
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Managing and searching for users](#)」を参照してください。

例 2: クライアント側のフィルターを使用してユーザーを一覧表示するには

次の list-users の例では、属性を持つ 3 人のユーザーの属性を一覧表示します。この場合は、E メールアドレスに、E メールドメイン「@example.com」が含まれています。他の属性

がこの文字列を含んでいる場合、その属性も表示されます。2 番目のユーザーは、クエリに一致する属性を持たず、表示される出力から除外されますが、サーバーのレスポンスからは除外されません。

```
aws cognito-idp list-users \
  --user-pool-id us-west-2_EXAMPLE \
  --max-items 3
  --query Users\[*\].Attributes\[?Value\.contains\(\@,\.'@example.com\.'\)\]
```

出力:

```
[
  [
    {
      "Name": "email",
      "Value": "admin@example.com"
    }
  ],
  [],
  [
    {
      "Name": "email",
      "Value": "operator@example.com"
    }
  ]
]
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Managing and searching for users](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListUsers](#)」を参照してください。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderException;
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);
    }
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[ListUsers](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
const listUsers = ({ userPoolId }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new ListUsersCommand({
        UserPoolId: userPoolId,
    });

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

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[ListUsers](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun listAllUsers(userPoolId: String) {
    val request =
        ListUsersRequest {
            this.userPoolId = userPoolId
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
    { cognitoClient ->
        val response = cognitoClient.listUsers(request)
        response.users?.forEach { user ->
            println("The user name is ${user.username}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「[ListUsers](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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

```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[ListUsers](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

TRY.
  DATA(lo_result) = lo_cgp->listusers(
    iv_userpoolid = iv_user_pool_id
  ).

  ot_users = lo_result->get_users( ).

  MESSAGE |Found { lines( ot_users ) } users in the pool.| TYPE 'I'.

  CATCH /aws1/cx_cgpresourcenotfoundex INTO DATA(lo_ex).
    MESSAGE |User pool { iv_user_pool_id } not found.| TYPE 'E'.

  CATCH /aws1/cx_cgpnnotauthorizedex INTO DATA(lo_auth_ex).
    MESSAGE 'Not authorized to list users.' TYPE 'E'.
ENDTRY.

```


- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「[ListUsers](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
do {
    let output = try await cognitoClient.listUsers(
        input: ListUsersInput(
            userPoolId: poolId
        )
    )

    guard let users = output.users else {
        print("No users found.")
        return
    }

    print("\(users.count) user(s) found.")
    for user in users {
        print("  \(user.username ?? "<unknown>")")
    }
} catch _ as NotAuthorizedException {
    print("**** Please authenticate with AWS before using this command.")
    return
} catch _ as ResourceNotFoundException {
    print("**** The specified User Pool was not found.")
    return
} catch {
    print("**** An unexpected type of error occurred.")
    return
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[ListUsers](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ResendConfirmationCode**で を使用する

次のサンプルコードは、ResendConfirmationCode を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[ResendConfirmationCode](#)」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
        }
    }
```

- APIの詳細については、「AWS SDK for C++ API リファレンス」の「[ResendConfirmationCode](#)」を参照してください。

CLI

AWS CLI

確認コードを再送信するには

次の `resend-confirmation-code` 例では、ユーザー `jane` に確認コードを送信します。

```
aws cognito-idp resend-confirmation-code \
  --client-id 12a3b456c7de890f11g123hijk \
  --username jane
```

出力:

```
{
  "CodeDeliveryDetails": {
    "Destination": "j***@e***.com",
    "DeliveryMedium": "EMAIL",
    "AttributeName": "email"
  }
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[ユーザーアカウントのサインアップと確認](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ResendConfirmationCode](#)」を参照してください。

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ResendConfirmationCode](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
const resendConfirmationCode = ({ clientId, username }) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new ResendConfirmationCodeCommand({  
    ClientId: clientId,  
    Username: username,  
  });  
  
  return client.send(command);  
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[ResendConfirmationCode](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
suspend fun resendConfirmationCode(  
  clientIdVal: String?,  
  userNameVal: String?,  
) {  
  val codeRequest =
```

```

        ResendConfirmationCodeRequest {
            clientId = clientIdVal
            username = userNameVal
        }

        CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
{ identityProviderClient ->
    val response = identityProviderClient.resendConfirmationCode(codeRequest)
    println("Method of delivery is " +
(response.codeDeliveryDetails?.deliveryMedium))
}
}

```

- APIの詳細については、AWS SDK for Kotlin API リファレンスの「[ResendConfirmationCode](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

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
```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[ResendConfirmationCode](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

/// Requests a new confirmation code be sent to the given user's contact
/// method.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - clientId: The application client ID.
///   - userName: The user to resend a code for.
///
/// - Returns: `true` if a new code was sent successfully, otherwise
///   `false`.
func resendConfirmationCode(cipClient: CognitoIdentityProviderClient,
                           clientId: String,
                           userName: String) async -> Bool {
    do {
        let output = try await cipClient.resendConfirmationCode(
            input: ResendConfirmationCodeInput(
                clientId: clientId,
                username: userName
            )
        )

        guard let deliveryMedium = output.codeDeliveryDetails?.deliveryMedium
    else {
        print("*** Unable to get the delivery method for the resent
code.")
        return false
    }

    print("=====> A new code has been sent by \(deliveryMedium)")
}
```

```
        return true
    } catch {
        print("*** Unable to resend the confirmation code to user
\\(userName).")
        return false
    }
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[ResendConfirmationCode](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **RespondToAuthChallenge**で を使用する

次のサンプルコードは、RespondToAuthChallenge を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

CLI

AWS CLI

例 1: NEW_PASSWORD_REQUIRED チャレンジに応答するには

次の respond-to-auth-challenge の例では、initiate-auth によって返された NEW_PASSWORD_REQUIRED チャレンジに応答します。ユーザー jane@example.com のパスワードを設定します。

```
aws cognito-idp respond-to-auth-challenge \
  --client-id 1example23456789 \
  --challenge-name NEW_PASSWORD_REQUIRED \
  --challenge-responses USERNAME=jane@example.com,NEW_PASSWORD=[Password] \
  --session AYABeEv5Hk1EXAMPLE
```

出力:

```
{
  "ChallengeParameters": {},
  "AuthenticationResult": {
    "AccessToken": "ACCESS_TOKEN",
    "ExpiresIn": 3600,
    "TokenType": "Bearer",
    "RefreshToken": "REFRESH_TOKEN",
    "IdToken": "ID_TOKEN",
    "NewDeviceMetadata": {
      "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
      "DeviceGroupKey": "-wt2ha1Zd"
    }
  }
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Authentication](#)」を参照してください。

例 2: SELECT_MFA_TYPE チャレンジに応答するには

次の respond-to-auth-challenge の例では、現在のユーザーの MFA オプションとして TOTP MFA を選択します。ユーザーは、MFA タイプを選択して MFA コードを入力するように求められます。

```
aws cognito-idp respond-to-auth-challenge \
  --client-id 1example23456789
  --session AYABeEv5Hk1EXAMPLE
  --challenge-name SELECT_MFA_TYPE
  --challenge-responses USERNAME=testuser,ANSWER=SOFTWARE_TOKEN_MFA
```

出力:

```
{
  "ChallengeName": "SOFTWARE_TOKEN_MFA",
  "Session": "AYABeEv5Hk1EXAMPLE",
  "ChallengeParameters": {
    "FRIENDLY_DEVICE_NAME": "transparent"
  }
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Adding MFA](#)」を参照してください。

例 3: SOFTWARE_TOKEN_MFA チャレンジに応答するには

次の respond-to-auth-challenge の例では、TOTP MFA コードを提供し、サインインを完了する方法を示します。

```
aws cognito-idp respond-to-auth-challenge \  
  --client-id 1example23456789 \  
  --session AYABeEv5Hk1EXAMPLE \  
  --challenge-name SOFTWARE_TOKEN_MFA \  
  --challenge-responses USERNAME=testuser,SOFTWARE_TOKEN_MFA_CODE=123456
```

出力:

```
{  
  "AuthenticationResult": {  
    "AccessToken": "eyJra456defEXAMPLE",  
    "ExpiresIn": 3600,  
    "TokenType": "Bearer",  
    "RefreshToken": "eyJra123abcEXAMPLE",  
    "IdToken": "eyJra789ghiEXAMPLE",  
    "NewDeviceMetadata": {  
      "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",  
      "DeviceGroupKey": "-v7w9UcY6"  
    }  
  }  
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Adding MFA](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[RespondToAuthChallenge](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[RespondToAuthChallenge](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

追跡対象デバイスでサインインします。サインインを完了するには、クライアントはセキュアリモートパスワード (SRP) チャレンジに正しく応答する必要があります。

```
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 {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

```


- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[RespondToAuthChallenge](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **SignUp**で を使用する

次のサンプルコードは、SignUp を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Lambda 関数を使用して登録済みのユーザーを自動的に確認する](#)
- [Lambda 関数を使用して登録済みのユーザーを自動的に移行する](#)
- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[SignUp](#)」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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().GetErrorType() ==

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;
    }
}

```

- APIの詳細については、「AWS SDK for C++ API リファレンス」の「[SignUp](#)」を参照してください。

CLI

AWS CLI

ユーザーをサインアップするには

この例では jane@example.com をサインアップします。

コマンド:

```
aws cognito-idp sign-up --client-id 3n4b5urk1ft4fl3mg5e62d9ado --  
username jane@example.com --password PASSWORD --user-attributes  
Name="email",Value="jane@example.com" Name="name",Value="Jane"
```

出力:

```
{  
  "UserConfirmed": false,  
  "UserSub": "e04d60a6-45dc-441c-a40b-e25a787d4862"  
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SignUp](#)」を参照してください。

Go

SDK for Go V2

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[SignUp](#)」を参照してください。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[SignUp](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[SignUp](#)」を参照してください。

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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.fromEnvironment { region = "us-east-1" }.use
    { identityProviderClient ->
        identityProviderClient.signUp(signUpRequest)
        println("User has been signed up")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[SignUp](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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

```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[SignUp](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

import AWSClientRuntime
import AWSCognitoIdentityProvider

```

```

    /// Create a new user in a user pool.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The ID of the app client to create a user for.
    ///   - userName: The username for the new user.
    ///   - password: The new user's password.
    ///   - email: The new user's email address.
    ///
    /// - Returns: `true` if successful; otherwise `false`.
    func signUp(cipClient: CognitoIdentityProviderClient, clientId: String,
        userName: String, password: String, email: String) async -> Bool {
        let emailAttr = CognitoIdentityProviderClientTypes.AttributeType(
            name: "email",
            value: email
        )

        let userAttrsList = [emailAttr]

        do {
            _ = try await cipClient.signUp(
                input: SignUpInput(
                    clientId: clientId,
                    password: password,
                    userAttributes: userAttrsList,
                    username: userName
                )
            )

            print("=====> User \(userName) signed up.")
        } catch _ as AWSognitoIdentityProvider.UsernameExistsException {
            print("*** The username \(userName) already exists. Please use a
different one.")
            return false
        } catch let error as AWSognitoIdentityProvider.InvalidPasswordException
        {
            print("*** Error: The specified password is invalid. Reason:
\((error.properties.message ?? "<none available>").")")
            return false
        } catch _ as AWSognitoIdentityProvider.ResourceNotFoundException {
            print("*** Error: The specified client ID (\(clientId)) doesn't
exist.")
            return false
        }
    }

```

```
    } catch {
        print("*** Unexpected error: \(error)")
        return false
    }

    return true
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[SignUp](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **UpdateUserPool** で を使用する

次のサンプルコードは、UpdateUserPool を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [Lambda 関数を使用して登録済みのユーザーを自動的に確認する](#)
- [Lambda 関数を使用して登録済みのユーザーを自動的に移行する](#)
- [Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

CLI

AWS CLI

ユーザープールを作成するには

次の update-user-pool の例では、使用可能な各設定オプションの構文例を使用して、ユーザープールを変更しています。ユーザープールを更新するには、以前に設定されたすべてのオプションを指定する必要があります。指定しない場合は、デフォルト値にリセットされます。

```
aws cognito-idp 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="amdin
+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=
\[\{Priority=1,Name="verified_email"\},
\{Priority=2,Name="verified_phone_number"\}\]

```

このコマンドでは何も出力されません。

詳細については、「Amazon Cognito Developer Guide」の「[Updating user pool configuration](#)」を参照してください。

- APIの詳細については、「AWS CLI コマンドリファレンス」の「[UpdateUserPool](#)」を参照してください。

Go

SDK for Go V2

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「[UpdateUserPool](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/**
 * 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];  
  }  
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[UpdateUserPool](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **VerifySoftwareToken**で を使用する

次のサンプルコードは、VerifySoftwareToken を使用する方法を説明しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- [MFA を必要とするユーザープールでユーザーをサインアップする](#)

.NET

SDK for .NET

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/// <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;
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[VerifySoftwareToken](#)」を参照してください。

C++

SDK for C++

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[VerifySoftwareToken](#)」を参照してください。

CLI

AWS CLI

TOTP Authenticator の登録を確認するには

次の `verify-software-token` の例では、現在のユーザーの TOTP 登録を完了します。

```
aws cognito-idp verify-software-token \
  --access-token eyJra456defEXAMPLE \
  --user-code 123456
```

出力:

```
{
  "Status": "SUCCESS"
}
```

詳細については、「Amazon Cognito デベロッパーガイド」の「[Adding MFA to a user pool](#)」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[VerifySoftwareToken](#)」を参照してください。

Java

SDK for Java 2.x

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// 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);
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[VerifySoftwareToken](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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);
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[VerifySoftwareToken](#)」を参照してください。

Kotlin

SDK for Kotlin

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
// Verify the TOTP and register for MFA.
suspend fun verifyTOTP(
    sessionVal: String?,
    codeVal: String?,
) {
    val tokenRequest =
        VerifySoftwareTokenRequest {
            userCode = codeVal
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
    { identityProviderClient ->
        val verifyResponse =
            identityProviderClient.verifySoftwareToken(tokenRequest)
        println("The status of the token is ${verifyResponse.status}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[VerifySoftwareToken](#)」を参照してください。

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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

```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[VerifySoftwareToken](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

TRY.

```

DATA(lo_result) = lo_cgp->verifysoftwaretoken(
    iv_session = iv_session
    iv_usercode = iv_user_code
).

```

```

ov_status = lo_result->get_status( ).

```

```

IF ov_status = 'SUCCESS'.

```

```

    MESSAGE 'MFA token verified successfully.' TYPE 'I'.

```

```

ELSE.

```

```

    MESSAGE |MFA verification status: { ov_status }.| TYPE 'I'.

```

```

ENDIF.

```



```

CATCH /aws1/cx_cgpcodemismatchex INTO DATA(lo_code_ex).
  MESSAGE 'Invalid MFA code provided.' TYPE 'E'.

CATCH /aws1/cx_cgpenbsoftwaretokmf00 INTO DATA(lo_enabled_ex).
  MESSAGE 'Software token MFA is already enabled.' TYPE 'E'.
ENDTRY.

```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの[VerifySoftwareToken](#)」を参照してください。

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

import AWSClientRuntime
import AWSCognitoIdentityProvider

/// Confirm that the user's TOTP authenticator is configured correctly by
/// sending a code to it to check that it matches successfully.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - session: An authentication session previously returned by an
///     `associateSoftwareToken()` call.
///   - mfaCode: The 6-digit code currently displayed by the user's
///     authenticator, as provided by the user.
func verifyTOTP(cipClient: CognitoIdentityProviderClient, session: String?,
mfaCode: String?) async {
    do {
        let output = try await cipClient.verifySoftwareToken(
            input: VerifySoftwareTokenInput(
                session: session,
                userCode: mfaCode
            )
        )
    }
}

```

```

    )

    guard let tokenStatus = output.status else {
        print("*** Unable to get the token's status.")
        return
    }
    print("=====> The token's status is: \(tokenStatus)")
} catch _ as SoftwareTokenMFANotFoundException {
    print("*** The specified user pool isn't configured for MFA.")
    return
} catch _ as CodeMismatchException {
    print("*** The specified MFA code doesn't match the expected value.")
    return
} catch _ as UserNotFoundException {
    print("*** The specified username doesn't exist.")
    return
} catch _ as UserNotConfirmedException {
    print("*** The user has not been confirmed.")
    return
} catch {
    print("*** Error verifying the MFA token!")
    return
}
}

```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[VerifySoftwareToken](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用する Amazon Cognito ID プロバイダーのシナリオ AWS SDKs

次のコード例は、AWS SDKs を使用して Amazon Cognito ID プロバイダーで一般的なシナリオを実装する方法を示しています。これらのシナリオは、Amazon Cognito ID プロバイダー内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出すことによって特定のタスクを実行する方法を示しています。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- [AWS SDK を使用して Lambda 関数で既知の Amazon Cognito ユーザーを自動的に確認する](#)
- [AWS SDK を使用して Lambda 関数を使用して既知の Amazon Cognito ユーザーを自動的に移行する](#)
- [AWS SDK を使用して MFA を必要とする Amazon Cognito ユーザープールでユーザーをサインアップする](#)
- [Amazon Cognito のアイデンティティプールと認証フローを使用する](#)
- [AWS SDK を使用して Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む](#)

AWS SDK を使用して Lambda 関数で既知の Amazon Cognito ユーザーを自動的に確認する

次のコード例は、Lambda 関数を使用して登録済みの Amazon Cognito ユーザーを自動的に確認する方法を示しています。

- PreSignUp トリガーの Lambda 関数を呼び出すようにユーザープールを設定します。
- Amazon Cognito でユーザーをサインアップする
- Lambda 関数は DynamoDB テーブルをスキャンし、登録済みのユーザーを自動的に確認します。
- 新しいユーザーとしてサインインし、リソースをクリーンアップします。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```

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))
}

```

Lambda 関数を使用して PreSignUp トリガーを処理します。

```

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)
}

```

一般的なタスクを実行する構造体を作成します。

```
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
    cwActor      *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))
}
```

Amazon Cognito アクションをラップする構造体を作成します。

```
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 {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
    return false, err
  }
  return true, nil
}

```

```
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
}
```

DynamoDB アクションをラップする構造体を作成します。

```
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
}
```

CloudWatch Logs アクションをラップする構造体を作成します。

```
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.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
}
```

CloudFormation アクションをラップする構造体を作成します。

```
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
}
```

リソースをクリーンアップします。

```
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!")  
}  
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [SignUp](#)
 - [UpdateUserPool](#)

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

インタラクティブな「シナリオ」実行を設定します。JavaScript (v3) の例では、シナリオランナーを共有して、複雑な例を効率化します。完全なソースコードは GitHub にあります。

```
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.",
  });
}

```

このシナリオでは、既知のユーザーを自動確認する方法を示します。サンプルのステップをオーケストレーションします。

```

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,
  );
```

これらは、他のシナリオと共有されるステップです。

```
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 },  
  );
```

Lambda 関数を使用する PreSignUp トリガーのハンドラー。

```
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);
};

```

CloudWatch Logs アクションのモジュール。

```
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];
  }
};

```

Amazon Cognito アクションのモジュール。

```

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").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        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];
  }
};
```

DynamoDB アクションのモジュール。

```
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];
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [SignUp](#)

- [UpdateUserPool](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して Lambda 関数を使用して既知の Amazon Cognito ユーザーを自動的に移行する

次のコード例は、Lambda 関数を使用して登録済みの Amazon Cognito ユーザーを自動的に移行する方法を示しています。

- MigrateUser トリガーの Lambda 関数を呼び出すようにユーザープールを設定します。
- ユーザープールにないユーザー名と E メールで Amazon Cognito にサインインします。
- Lambda 関数は DynamoDB テーブルをスキャンし、登録済みのユーザーをユーザープールに自動的に移行します。
- パスワードを忘れた場合のフローを実行して、移行したユーザーのパスワードをリセットします。
- 新しいユーザーとしてサインインし、リソースをクリーンアップします。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
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
        PasswordResetRequiredException.\n\n", user.UserName)
        authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
        if err != nil {
            if errors.As(err, &resetRequired) {
                log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
                DynamoDB known users table.\n"+
                    "User migration is started and a password reset is required.",
                    user.UserName)
            } else {

```

```

        panic(err)
    }
} else {
    log.Printf("User '%v' successfully signed in. This is unexpected and probably
means you have not\n"+
        "cleaned up a previous run of this scenario, so the user exist in the Cognito
user pool.\n"+
        "You can continue this example and select to clean up resources, or manually
remove\n"+
        "the user from your user pool and try again.", user.UserName)
    runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)
    signedIn = true
}
}

log.Println(strings.Repeat("-", 88))
return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string,
user actions.User) {
    wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user
to Cognito, you must be able to receive a confirmation\n"+
        "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
    if !wantCode {
        log.Println("To complete this example and successfully migrate a user to
Cognito, you must enter an email\n" +
            "you own that can receive a confirmation code.")
        return
    }
    codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
user.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))
}

```

Lambda 関数を使用して MigrateUser トリガーを処理します。

```

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)
}

```

一般的なタスクを実行する構造体を作成します。

```
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
    cwActor      *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))
}
```

Amazon Cognito アクションをラップする構造体を作成します。

```
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 {
    log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
  return confirmed, err
}

```



```

    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
}
```

DynamoDB アクションをラップする構造体を作成します。

```
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
}
```

CloudWatch Logs アクションをラップする構造体を作成します。

```
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.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
}
```

CloudFormation アクションをラップする構造体を作成します。

```
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
}
```

リソースをクリーンアップします。

```
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!")  
}  
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
 - [ConfirmForgotPassword](#)
 - [DeleteUser](#)
 - [ForgotPassword](#)
 - [InitiateAuth](#)
 - [SignUp](#)
 - [UpdateUserPool](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して MFA を必要とする Amazon Cognito ユーザープールでユーザーをサインアップする

次のコード例は、以下を実行する方法を示しています。

- ユーザー名、パスワード、E メールアドレスでサインアップしてユーザーを確認します。
- MFA アプリケーションをユーザーに関連付けて、多要素認証を設定します。
- パスワードと MFA コードを使用してサインインします。

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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.WriteLine("Email: ");
        email = Console.ReadLine();
    } while (string.IsNullOrEmpty(email));
}

// Now sign up the user.
Console.WriteLine($"{Environment.NewLine}Signing 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($"Confirmation code sent to {userName}.");
Console.WriteLine("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.WriteLine("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.Paginators.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.Paginators.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>
    /// 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>
    /// 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)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(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;
}
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。

- [AdminGetUser](#)
- [AdminInitiateAuth](#)
- [AdminRespondToAuthChallenge](#)
- [AssociateSoftwareToken](#)
- [ConfirmDevice](#)
- [ConfirmSignUp](#)
- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

C++

SDK for C++

 Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```

    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().GetErrorType() ==
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;
#ifdef 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::AdminGetUserRequest request;
    request.SetUsername(userName);
    request.SetUserPoolId(userPoolID);

    Aws::CognitoIdentityProvider::Model::AdminGetUserOutcome outcome =
        client.AdminGetUser(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::AdminGetUser. "
            << 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();
}

```


- API の詳細については、「AWS SDK for C++ API リファレンス」の以下のトピックを参照してください。
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import
    software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminGetUserRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminGetUserResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminInitiateAuthRequest;
```

```
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminInitiateAuthResponse;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminRespondToAuthChalleng
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AdminRespondToAuthChalleng
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AssociateSoftwareTokenRequ
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.AssociateSoftwareTokenResp
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.CognitoIdentityProviderExc
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ConfirmSignUpRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ResendConfirmationCodeRequ
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.ResendConfirmationCodeResp
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.SignUpRequest;
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.VerifySoftwareTokenRequest
import
    software.amazon.awssdk.services.cognitoidentityprovider.model.VerifySoftwareTokenRespons
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 {
            AdminGetUserRequest userRequest = AdminGetUserRequest.builder()
                .username(userName)
                .userPoolId(poolId)
                .build();

            AdminGetUserResponse response =
identityProviderClient.adminGetUser(userRequest);
            System.out.println("User status " + response.userStatusAsString());

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

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

最適なエクスペリエンスを得るには、GitHub リポジトリを複製してこの例を実行します。次のコードは、サンプルアプリケーション全体のサンプルを表しています。

```
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);
  } /**
```

```

    * @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);
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Kotlin

SDK for Kotlin

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
/**
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 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 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 = in0b.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 = in0b.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.fromEnvironment { 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.fromEnvironment { 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.fromEnvironment { 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.fromEnvironment { 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.fromEnvironment { 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.fromEnvironment { region = "us-east-1" }.use
        { identityProviderClient ->
            identityProviderClient.confirmSignUp(signUpRequest)
            println("$userNameVal was confirmed")
        }
    }

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

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
    { identityProviderClient ->
        val response = identityProviderClient.adminGetUser(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.fromEnvironment { region = "us-east-1" }.use
{ identityProviderClient ->
    identityProviderClient.signUp(signUpRequest)
    println("User has been signed up")
}
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の以下のトピックを参照してください。
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)
 - [ConfirmDevice](#)
 - [ConfirmSignUp](#)
 - [InitiateAuth](#)
 - [ListUsers](#)
 - [ResendConfirmationCode](#)
 - [RespondToAuthChallenge](#)
 - [SignUp](#)
 - [VerifySoftwareToken](#)

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

シナリオで使用される Amazon Cognito 関数をラップするクラスを作成します。

```
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 {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_auth['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

```

シナリオを実行するクラスを作成します。この例では Amazon Cognito で追跡する MFA デバイスを登録し、追跡対象デバイスのパスワードと情報を使用してサインインする方法も示します。これにより、新しい MFA コードを入力する必要がなくなります。

```

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-idp"), args.user_pool_id,
args.client_id)
    except Exception:
        logging.exception("Something went wrong with the demo.")

if __name__ == "__main__":
    main()
```

- APIの詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。

- [AdminGetUser](#)
- [AdminInitiateAuth](#)
- [AdminRespondToAuthChallenge](#)
- [AssociateSoftwareToken](#)
- [ConfirmDevice](#)
- [ConfirmSignUp](#)
- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

Swift

SDK for Swift

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

Package.swift ファイル。

```
// swift-tools-version: 5.9
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "cognito-scenario",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/aws-labs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
        // or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "cognito-scenario",
```

```
        dependencies: [
            .product(name: "AWSCognitoIdentityProvider", package: "aws-sdk-swift"),
            .product(name: "ArgumentParser", package: "swift-argument-parser")
        ],
        path: "Sources")
    ]
}
```

Swift コードファイル。

```
// An example demonstrating various features of Amazon Cognito. Before running
// this Swift 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 example performs the following functions:
//
// 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 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.
```

```
import ArgumentParser
import Foundation

import AWSClientRuntime
import AWSCognitoIdentityProvider

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The application clientId.")
    var clientId: String
    @Argument(help: "The user pool ID to use.")
    var poolId: String
    @Option(help: "Name of the Amazon Region to use")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "cognito-scenario",
        abstract: ""
        Demonstrates various features of Amazon Cognito.
        "",
        discussion: ""
        ""
    )

    /// Prompt for an input string of at least a minimum length.
    ///
    /// - Parameters:
    ///   - prompt: The prompt string to display.
    ///   - minLength: The minimum number of characters to allow in the
    ///     response. Default value is 0.
    ///
    /// - Returns: The entered string.
    func stringRequest(_ prompt: String, minLength: Int = 1) -> String {
        while true {
            print(prompt, terminator: "")
            let str = readLine()

            guard let str else {
                continue
            }
            if str.count >= minLength {
                return str
            } else {
```

```
        print("*** Response must be at least \$(minLength) character(s)
long.")
    }
}

/// Ask a yes/no question.
///
/// - Parameter prompt: A prompt string to print.
///
/// - Returns: `true` if the user answered "Y", otherwise `false`.
func yesNoRequest(_ prompt: String) -> Bool {
    while true {
        let answer = stringRequest(prompt).lowercased()
        if answer == "y" || answer == "n" {
            return answer == "y"
        }
    }
}

/// Get information about a specific user in a user pool.
///
/// - Parameters:
///   - cipClient: The Amazon Cognito Identity Provider client to use.
///   - userName: The user to retrieve information about.
///   - userPoolId: The user pool to search for the specified user.
///
/// - Returns: `true` if the user's information was successfully
///   retrieved. Otherwise returns `false`.
func adminGetUser(cipClient: CognitoIdentityProviderClient, userName: String,
                  userPoolId: String) async -> Bool {
    do {
        let output = try await cipClient.adminGetUser(
            input: AdminGetUserInput(
                userPoolId: userPoolId,
                username: userName
            )
        )

        guard let userStatus = output.userStatus else {
            print("*** Unable to get the user's status.")
            return false
        }
    }
}
```

```

        print("User status: \(userStatus)")
        return true
    } catch {
        return false
    }
}

/// Create a new user in a user pool.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - clientId: The ID of the app client to create a user for.
///   - userName: The username for the new user.
///   - password: The new user's password.
///   - email: The new user's email address.
///
/// - Returns: `true` if successful; otherwise `false`.
func signUp(cipClient: CognitoIdentityProviderClient, clientId: String,
userName: String, password: String, email: String) async -> Bool {
    let emailAttr = CognitoIdentityProviderClientTypes.AttributeType(
        name: "email",
        value: email
    )

    let userAttrsList = [emailAttr]

    do {
        _ = try await cipClient.signUp(
            input: SignUpInput(
                clientId: clientId,
                password: password,
                userAttributes: userAttrsList,
                username: userName
            )
        )

        print("=====> User \(userName) signed up.")
    } catch _ as AWSognitoIdentityProvider.UsernameExistsException {
        print("*** The username \(userName) already exists. Please use a
different one.")
        return false
    } catch let error as AWSognitoIdentityProvider.InvalidPasswordException
{

```

```

        print("*** Error: The specified password is invalid. Reason:
\\(error.properties.message ?? "<none available>").")
        return false
    } catch _ as AWSIdentityProvider.ResourceNotFoundException {
        print("*** Error: The specified client ID (\\(clientId)) doesn't
exist.")
        return false
    } catch {
        print("*** Unexpected error: \\(error)")
        return false
    }

    return true
}

/// Requests a new confirmation code be sent to the given user's contact
/// method.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - clientId: The application client ID.
///   - userName: The user to resend a code for.
///
/// - Returns: `true` if a new code was sent successfully, otherwise
///   `false`.
func resendConfirmationCode(cipClient: CognitoIdentityProviderClient,
clientId: String,
                           userName: String) async -> Bool {
    do {
        let output = try await cipClient.resendConfirmationCode(
            input: ResendConfirmationCodeInput(
                clientId: clientId,
                username: userName
            )
        )

        guard let deliveryMedium = output.codeDeliveryDetails?.deliveryMedium
    else {
        print("*** Unable to get the delivery method for the resent
code.")
        return false
    }

    print("=====> A new code has been sent by \\(deliveryMedium)")

```



```

        return true
    } catch {
        print("*** Unable to resend the confirmation code to user
\\(userName).")
        return false
    }
}

/// Submit a confirmation code for the specified user. This is the code as
/// entered by the user after they've received it by email or text
/// message.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - clientId: The app client ID the user is signing up for.
///   - userName: The username of the user whose code is being sent.
///   - code: The user's confirmation code.
///
/// - Returns: `true` if the code was successfully confirmed; otherwise
`false`.
func confirmSignUp(cipClient: CognitoIdentityProviderClient, clientId:
String,
                  userName: String, code: String) async -> Bool {
    do {
        _ = try await cipClient.confirmSignUp(
            input: ConfirmSignUpInput(
                clientId: clientId,
                confirmationCode: code,
                username: userName
            )
        )

        print("=====> \\(userName) has been confirmed.")
        return true
    } catch {
        print("=====> \\(userName)'s code was entered incorrectly.")
        return false
    }
}

/// Begin an authentication session.
///
/// - Parameters:
///   - cipClient: The `CongitoIdentityProviderClient` to use.

```

```

/// - clientId: The app client ID to use.
/// - userName: The username to check.
/// - password: The user's password.
/// - userPoolId: The user pool to use.
///
/// - Returns: The session token associated with this authentication
/// session.
func initiateAuth(cipClient: CognitoIdentityProviderClient, clientId: String,
                  userName: String, password: String,
                  userPoolId: String) async -> String? {
    var authParams: [String: String] = [:]

    authParams["USERNAME"] = userName
    authParams["PASSWORD"] = password

    do {
        let output = try await cipClient.adminInitiateAuth(
            input: AdminInitiateAuthInput(
                authFlow:
CognitoIdentityProviderClientTypes.AuthFlowType.adminUserPasswordAuth,
                authParameters: authParams,
                clientId: clientId,
                userPoolId: userPoolId
            )
        )

        guard let challengeName = output.challengeName else {
            print("*** Invalid response from the auth service.")
            return nil
        }

        print("=====> Response challenge is \(challengeName)")

        return output.session
    } catch _ as UserNotFoundException {
        print("*** The specified username, \(userName), doesn't exist.")
        return nil
    } catch _ as UserNotConfirmedException {
        print("*** The user \(userName) has not been confirmed.")
        return nil
    } catch {
        print("*** An unexpected error occurred.")
        return nil
    }
}

```

```

    }

    /// Request and display an MFA secret token that the user should enter
    /// into their authenticator to set it up for the user account.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - authSession: The authentication session to request an MFA secret
    ///     for.
    ///
    /// - Returns: A string containing the MFA secret token that should be
    ///   entered into the authenticator software.
    func getSecretForAppMFA(cipClient: CognitoIdentityProviderClient,
    authSession: String?) async -> String? {
        do {
            let output = try await cipClient.associateSoftwareToken(
                input: AssociateSoftwareTokenInput(
                    session: authSession
                )
            )

            guard let secretCode = output.secretCode else {
                print("*** Unable to get the secret code")
                return nil
            }

            print("=====> Enter this token into Google Authenticator:
            \(secretCode)")
            return output.session
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return nil
        } catch {
            print("*** An unexpected error occurred getting the secret for the
            app's MFA.")
            return nil
        }
    }

    /// Confirm that the user's TOTP authenticator is configured correctly by
    /// sending a code to it to check that it matches successfully.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.

```

```

    /// - session: An authentication session previously returned by an
    ///       `associateSoftwareToken()` call.
    /// - mfaCode: The 6-digit code currently displayed by the user's
    ///       authenticator, as provided by the user.
    func verifyTOTP(cipClient: CognitoIdentityProviderClient, session: String?,
    mfaCode: String?) async {
        do {
            let output = try await cipClient.verifySoftwareToken(
                input: VerifySoftwareTokenInput(
                    session: session,
                    userCode: mfaCode
                )
            )

            guard let tokenStatus = output.status else {
                print("*** Unable to get the token's status.")
                return
            }
            print("=====> The token's status is: \(tokenStatus)")
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return
        } catch _ as CodeMismatchException {
            print("*** The specified MFA code doesn't match the expected value.")
            return
        } catch _ as UserNotFoundException {
            print("*** The specified username doesn't exist.")
            return
        } catch _ as UserNotConfirmedException {
            print("*** The user has not been confirmed.")
            return
        } catch {
            print("*** Error verifying the MFA token!")
            return
        }
    }
}

/// Respond to the authentication challenge received from Cognito after
/// initiating an authentication session. This involves sending a current
/// MFA code to the service.
///
/// - Parameters:
///   - cipClient: The `CognitoIdentityProviderClient` to use.
///   - userName: The user's username.

```

```

/// - clientId: The app client ID.
/// - userPoolId: The user pool to sign into.
/// - mfaCode: The 6-digit MFA code currently displayed by the user's
///   authenticator.
/// - session: The authentication session to continue processing.
func adminRespondToAuthChallenge(cipClient: CognitoIdentityProviderClient,
    userName: String,
                                clientId: String, userPoolId: String,
    mfaCode: String,
                                session: String) async {
    print("=====> SOFTWARE_TOKEN_MFA challenge is generated...")

    var challengeResponsesOb: [String: String] = [:]
    challengeResponsesOb["USERNAME"] = userName
    challengeResponsesOb["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

    do {
        let output = try await cipClient.adminRespondToAuthChallenge(
            input: AdminRespondToAuthChallengeInput(
                challengeName:
CognitoIdentityProviderClientTypes.ChallengeNameType.softwareTokenMfa,
                challengeResponses: challengeResponsesOb,
                clientId: clientId,
                session: session,
                userPoolId: userPoolId
            )
        )

        guard let authenticationResult = output.authenticationResult else {
            print("*** Unable to get authentication result.")
            return
        }

        print("=====> Authentication result (JWTs are redacted):")
        print(authenticationResult)
    } catch _ as SoftwareTokenMFANotFoundException {
        print("*** The specified user pool isn't configured for MFA.")
        return
    } catch _ as CodeMismatchException {
        print("*** The specified MFA code doesn't match the expected value.")
        return
    } catch _ as UserNotFoundException {
        print("*** The specified username, \(userName), doesn't exist.")
        return
    }
}

```

```

    } catch _ as UserNotConfirmedException {
        print("*** The user \$(userName) has not been confirmed.")
        return
    } catch let error as NotAuthorizedException {
        print("*** Unauthorized access. Reason: \$(error.properties.message ??
"<unknown>")")
    } catch {
        print("*** Error responding to the MFA challenge.")
        return
    }
}

/// Called by ``main()`` to run the bulk of the example.
func runAsync() async throws {
    let config = try await
CognitoIdentityProviderClient.CognitoIdentityProviderClientConfiguration(region:
region)
    let cipClient = CognitoIdentityProviderClient(config: config)

    print("""
        This example collects information about a user, then creates that
user in the
        specified user pool. Then, it enables Multi-Factor Authentication
(MFA) for that
        user by associating an authenticator application (such as Google
Authenticator
        or a password manager that supports TOTP). Then, the user uses a
code from their
        authenticator application to sign in.

        """)

    let userName = stringRequest("Please enter a new username: ")
    let password = stringRequest("Enter a password: ")
    let email = stringRequest("Enter your email address: ", minLength: 5)

    // Submit the sign-up request to AWS.

    print("==> Signing up user \$(userName)...")
    if await signUp(cipClient: cipClient, clientId: clientId,
                    userName: userName, password: password,
                    email: email) == false {
        return
    }
}

```

```
// Check the user's status. This time, it should come back "unconfirmed".

print("==> Getting the status of user \(userName) from the user pool
(should be 'unconfirmed')...")
if await adminGetUser(cipClient: cipClient, userName: userName,
userPoolId: poolId) == false {
    return
}

// Ask the user if they want a replacement code sent, such as if the
// code hasn't arrived yet. If the user responds with a "yes," send a
// new code.

if yesNoRequest("==> A confirmation code was sent to \(userName). Would
you like to send a new code (Y/N)? ") {
    print("==> Sending a new confirmation code...")
    if await resendConfirmationCode(cipClient: cipClient, clientId:
clientId, userName: userName) == false {
        return
    }
}

// Ask the user to enter the confirmation code, then send it to Amazon
// Cognito to verify it.

let code = stringRequest("==> Enter the confirmation code sent to
\(userName): ")
if await confirmSignUp(cipClient: cipClient, clientId: clientId,
userName: userName, code: code) == false {
    // The code didn't match. Your application may wish to offer to
    // re-send the confirmation code here and try again.
    return
}

// Check the user's status again. This time it should come back
// "confirmed".

print("==> Rechecking status of user \(userName) in the user pool (should
be 'confirmed')...")
if await adminGetUser(cipClient: cipClient, userName: userName,
userPoolId: poolId) == false {
    return
}
```

```
// Check the challenge mode. Here, it should be "mfaSetup", indicating
// that the user needs to add MFA before using it. This returns a
// session that can be used to register MFA, or nil if an error occurs.

let authSession = await initiateAuth(cipClient: cipClient, clientId:
clientId,
                                userName: userName, password:
password,
                                userPoolId: poolId)

if authSession == nil {
    return
}

// Ask Cognito for an MFA secret token that the user should enter into
// their authenticator software (such as Google Authenticator) or
// password manager to configure it for this user account. This
// returns a new session that should be used for the new stage of the
// authentication process.

let newSession = await getSecretForAppMFA(cipClient: cipClient,
authSession: authSession)
if newSession == nil {
    return
}

// Ask the user to enter the current 6-digit code displayed by their
// authenticator. Then verify that it matches the value expected for
// the session.

let mfaCode1 = stringRequest("==> Enter the 6-digit code displayed in
your authenticator: ",
                                minLength: 6)
await verifyTOTP(cipClient: cipClient, session: newSession, mfaCode:
mfaCode1)

// Ask the user to authenticate now that the authenticator has been
// configured. This creates a new session using the user's username
// and password as already entered.

print("\nNow starting the sign-in process for user \(userName)...\n")

let session2 = await initiateAuth(cipClient: cipClient, clientId:
clientId,
```



```

        userName: userName, password: password,
        userPoolId: poolId)
        guard let session2 else {
            return
        }

        // Now that we have a new auth session, `session2`, ask the user for a
        // new 6-digit code from their authenticator, and send it to the auth
        // session.

        let mfaCode2 = stringRequest("==> Wait for your authenticator to show a
        new 6-digit code, then enter it: ",
                                    minLength: 6)

        await adminRespondToAuthChallenge(cipClient: cipClient, userName:
        userName,
                                    clientId: clientId, userPoolId: poolId,
                                    mfaCode: mfaCode2, session: session2)
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}

```

- APIの詳細については、AWS SDK for Swift API リファレンスの以下のトピックを参照してください。
 - [AdminGetUser](#)
 - [AdminInitiateAuth](#)
 - [AdminRespondToAuthChallenge](#)
 - [AssociateSoftwareToken](#)

- [ConfirmDevice](#)
- [ConfirmSignUp](#)
- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Cognito のアイデンティティプールと認証フローを使用する

次のコード例は、アイデンティティプールの認証フローを示すウェブベースのデモアプリケーションを作成する方法を示しています。

Python

SDK for Python (Boto3)

Amazon Cognito アイデンティティプールの認証フローを示すウェブベースのデモアプリケーションを通じて、ユーザーがさまざまな ID プロバイダーで拡張認証フローと基本認証フローの両方をインタラクティブに参照できるようにします。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID プロバイダー

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む

次のコード例は、Amazon Cognito ユーザー認証後に Lambda 関数を使用してカスタムアクティビティデータを書き込む方法を示しています。

- 管理者関数を使用して、ユーザーをユーザープールに追加します。
- PostAuthentication トリガーの Lambda 関数を呼び出すようにユーザープールを設定します。
- 新しいユーザーを Amazon Cognito にサインインします。
- Lambda 関数は、カスタム情報を CloudWatch Logs と DynamoDB テーブルに書き込みます。
- DynamoDB テーブルからカスタムデータを取得して表示し、リソースをクリーンアップします。

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
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))
}
```

Lambda 関数を使用して PostAuthentication トリガーを処理します。

```
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)
}

```

一般的なタスクを実行する構造体を作成します。

```

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))  
}
```

Amazon Cognito アクションをラップする構造体を作成します。

```
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
}
```

DynamoDB アクションをラップする構造体を作成します。

```
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
    Email 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), Email: 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
}
```

CloudWatch Logs アクションをラップする構造体を作成します。

```
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.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
}

```

CloudFormation アクションをラップする構造体を作成します。

```

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
}

```

リソースをクリーンアップします。

```

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!")
}
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
 - [AdminCreateUser](#)
 - [AdminSetUserPassword](#)
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [UpdateUserPool](#)

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Cognito Sync のコード例 AWS SDKs

次のコード例は、AWS Software Development Kit (SDK) で Amazon Cognito Sync を使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Cognito Sync

- [SDK を使用した Amazon Cognito Sync の基本的な例 AWS SDKs](#)
 - [SDK を使用した Amazon Cognito Sync のアクション AWS SDKs](#)
 - [AWS SDK ListIdentityPoolUsageで を使用する](#)

SDK を使用した Amazon Cognito Sync の基本的な例 AWS SDKs

次のコード例では、Amazon Cognito Sync を AWS SDK と使用する基本的な方法を示しています。

例

- [SDK を使用した Amazon Cognito Sync のアクション AWS SDKs](#)
 - [AWS SDK ListIdentityPoolUsageで を使用する](#)

SDK を使用した Amazon Cognito Sync のアクション AWS SDKs

次のコード例は、AWS SDKs で個々の Amazon Cognito Sync アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細なリストについては、[Amazon Cognito Sync API Reference](#) (Amazon Cognito Sync API リファレンス) を参照してください。

例

- [AWS SDK ListIdentityPoolUsageで を使用する](#)

AWS SDK `ListIdentityPoolUsage`で を使用する

次の例は、`ListIdentityPoolUsage` を使用方法を説明しています。

Rust

SDK for Rust

Note

GitHub には、その他のリソースもあります。用例一覧を検索し、[AWS コード例リポジトリ](#)での設定と実行の方法を確認してください。

```
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(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「[ListIdentityPoolUsage](#)」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してください[AWS SDK でのこのサービスの使用](#)。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

マルチテナントアプリケーションのベストプラクティス

Amazon Cognito ユーザープールは、Amazon Cognito クォータ内に収める必要がある大量のリクエストを生成するマルチテナントアプリケーションで動作します。顧客ベースが拡大したときにこの容量をスケールアップするには、[追加のクォータ容量を購入](#)できます。

Note

Amazon Cognito [クォータ](#)は、AWS アカウント および ごとに適用されます AWS リージョン。これらのクォータは、アプリケーション内のすべてのテナント間で共有されます。Amazon Cognito サービスクォータをチェックして、クォータがアプリケーションの想定されるボリュームとテナント数に対応できることを確認してください。

このセクションでは、同じリージョン および 内の Amazon Cognito リソース間でテナントを分離するために実装できる方法について説明します AWS アカウント。テナントを複数の AWS アカウントまたはリージョンに分割し、それぞれに独自のクォータを与えることもできます。マルチリージョンマルチテナンシーのその他のメリットには、可能な限り高い分離レベル、グローバルに分散されたユーザーに対する最短ネットワーク転送時間、組織の既存のディストリビューションモデルへの準拠などがあります。

単一リージョンのマルチテナンシーは、顧客や管理者にとってもメリットがあります。

次のリストは、共有リソースを使用したマルチテナンシーのメリットの一部を示しています。

マルチテナンシーのメリット

共通ユーザーディレクトリ

マルチテナンシーは、顧客が複数のアプリケーションにアカウントを持つモデルをサポートします。[サードパーティープロバイダーの ID をリンク](#)する先を 1 つの一貫したユーザープールプロファイルすることができます。ユーザープロファイルがテナントに固有である場合、単一のユーザープールを持つマルチテナンシー戦略では、ユーザー管理へのエントリポイントは 1 つになります。

共通セキュリティ

共有ユーザープールでは、単一のセキュリティ標準を作成し、同じ[脅威保護](#)、[多要素認証](#) (MFA)、および [AWS WAF](#) 標準をすべてのテナントに適用できます。AWS WAF ウェブ ACL は

関連付けるリソース AWS リージョン と同じ に存在する必要があるため、マルチテナンシーは複雑なリソースへの共有アクセスを提供します。マルチリージョン Amazon Cognito アプリケーションで一貫したセキュリティ設定を維持する場合は、リソース間で設定を複製する運用標準を適用する必要があります。

共通カスタマイズ

ユーザープールと ID プールは でカスタマイズできます AWS Lambda。ユーザープールでの [Lambda トリガー](#) の設定や、アイデンティティプールでの [Amazon Cognito イベント](#) の設定は、複雑になる可能性があります。Lambda 関数は、ユーザープールまたは ID プール AWS リージョン と同じ にある必要があります。共有 Lambda 関数は、カスタム認証フロー、ユーザー移行、トークン生成、リージョン内の他の関数について標準を適用できます。

共通メッセージング

Amazon Simple Notification Service (Amazon SNS) では、ユーザーに [SMS メッセージ](#) を送信する前に、特定のリージョンでの追加の設定が必要になります。特定のリージョン内に含まれる、Amazon Simple Email Service (Amazon SES) で検証済みの ID とドメインを使用して [E メールメッセージ](#) を送信できます。

マルチテナンシーを使用すると、この設定とメンテナンスのオーバーヘッドをすべてのテナント間で共有できます。Amazon SNS と Amazon SES はすべての AWS リージョンで利用できるわけではないため、リソースをリージョン間で分割するには追加の考慮事項が必要になります。

[カスタムメッセージングプロバイダー](#) を使用すると、メッセージ配信の管理に単一の Lambda 関数を使用し、共通カスタマイズを適用できます。

[\[マネージドログイン\]](#) は、ブラウザにセッション Cookie を設定し、認証済みのユーザーを認識します。ユーザープール内のローカルユーザーを認証する場合、ローカルユーザーのセッション Cookie は、同じユーザープール内のすべてのアプリケーションクライアントに対してユーザーを認証します。ローカルユーザーは、外部 IdP を介したフェデレーションなしに、ユーザープールディレクトリにのみ存在します。セッション Cookie は 1 時間有効です。セッション Cookie の有効期間を変更することはできません。

ホストされた UI セッション Cookie を使用した複数のアプリケーションクライアント間でのサインインを防ぐには、2 つの方法があります。

- ユーザーをテナントごとのユーザープールに分割します。
- ホストされた UI サインインを Amazon Cognito ユーザープール API サインインに置き換えます。

トピック

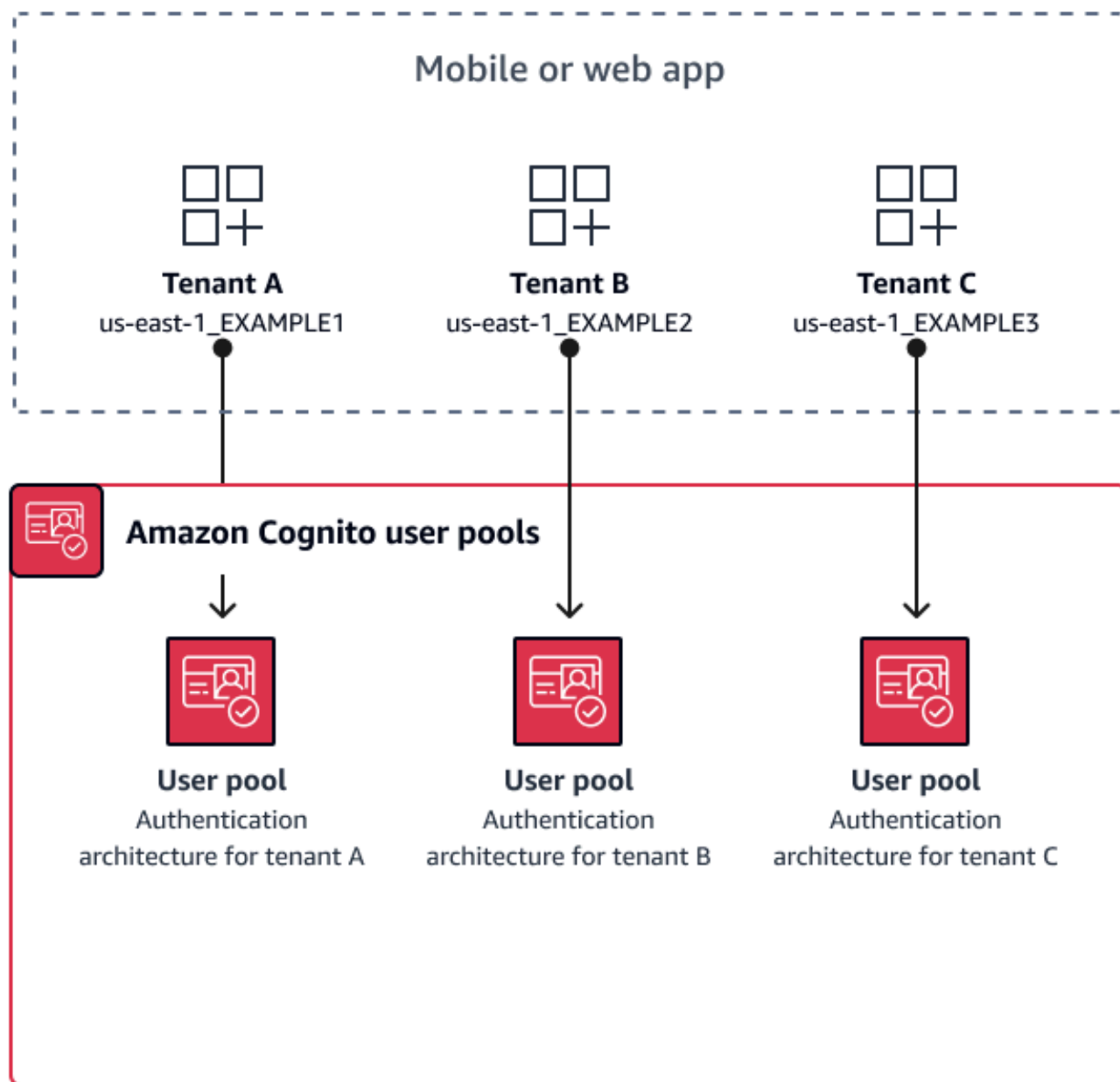
- [ユーザープールマルチテナンシーのベストプラクティス](#)
- [アプリケーションクライアントマルチテナンシーのベストプラクティス](#)
- [ユーザーグループのマルチテナンシーのベストプラクティス](#)
- [カスタム属性マルチテナンシーのベストプラクティス](#)
- [カスタムスコープマルチテナンシーのベストプラクティス](#)
- [マルチテナンシーのセキュリティに関する推奨事項](#)

ユーザープールマルチテナンシーのベストプラクティス

アプリケーション内のテナントごとにユーザープールを作成します。このアプローチは各テナントを最大限に分離します。テナントごとに異なる設定を実装することができます。ユーザープールによるテナントの分離は、ユーザー対テナントのマッピングに柔軟性を与えます。同じユーザーに複数のプロファイルを作成することができます。ただし、ユーザーはそれぞれ、アクセスできるテナントごとに個別にサインアップする必要があります。

このアプローチを使用すると、テナントごとにホストされた UI を個別にセットアップし、アプリケーションのテナント固有のインスタンスにユーザーをリダイレクトできます。このアプローチを使用して、[Amazon API Gateway](#) などのバックエンドサービスと統合することもできます。

次の図は、専用ユーザープールを持つ各テナントを示しています。



ユーザープールマルチテナンシーを実装するタイミング

分離とカスタマイズが主な懸念事項であるとき。複数のユーザープールを持つ単一のアーキテクチャでは、ユーザーとテナントの関係が複雑になる場合があります。教育関連テナントが2つある例を考えてみましょう。同じユーザーでも、あるアプリケーションではアクセスが制限された学生であるが、別のアプリケーションでは上位レベルのアクセス許可を持つ教師である場合があります。あるアプリケーションでは MFA を要求しているが別のアプリケーションでは要求しない場合や、パスワー

ドポリシーが異なる場合もあります。ローカルユーザーは、マネージドログインを使用してユーザープール内の複数のアプリケーションクライアントにサインインできるため、複数のテナントがマネージドログインを使用してサインインできるようにする場合にも、ユーザープールのマルチテナンシーが最適です。

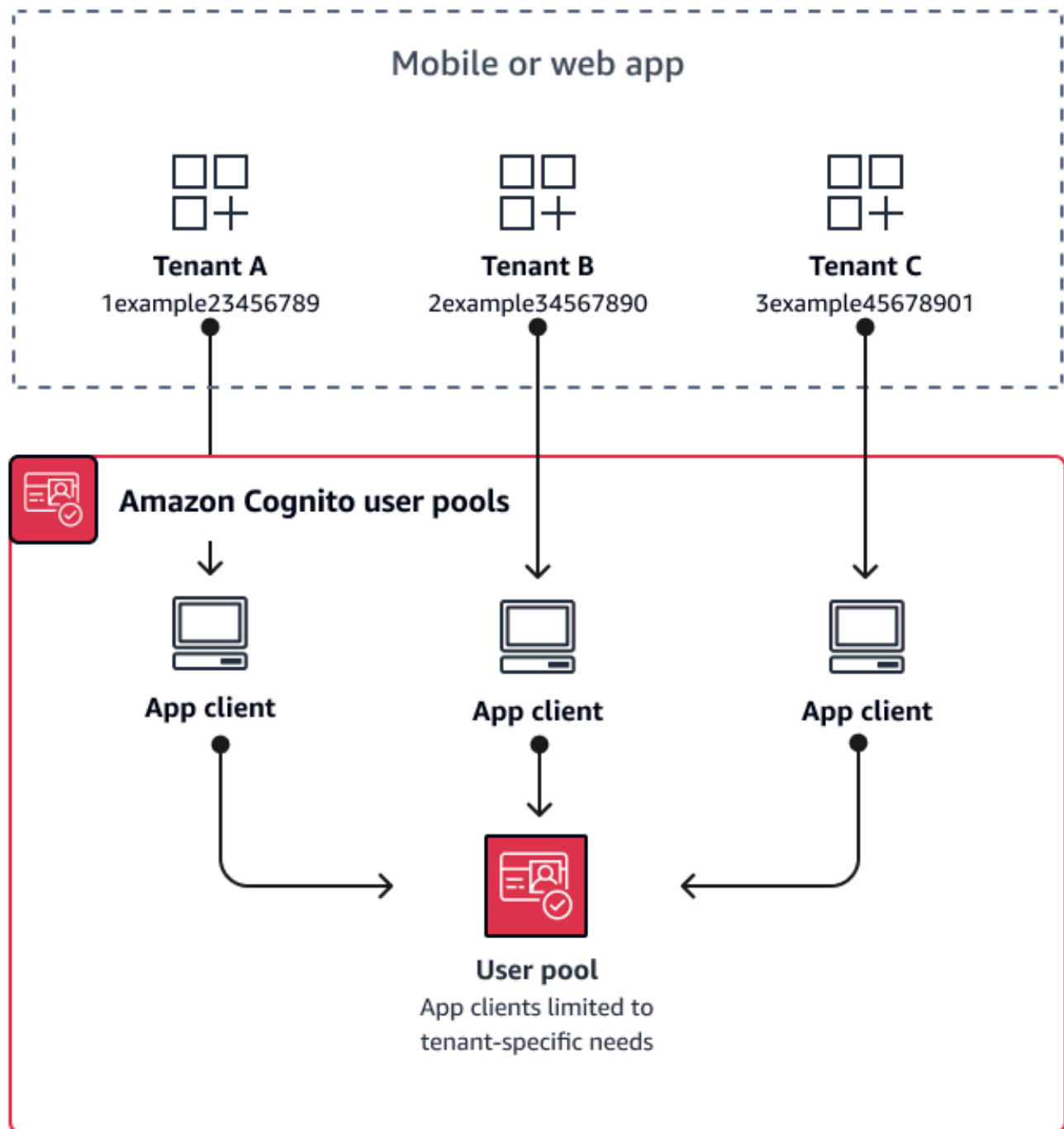
労力レベル

このアプローチを使用するには、高い開発労力と運用労力がかかります。アプリケーション群全体で一貫性のある予測可能な結果を確保するには、Amazon Cognito リソースをオートメーションツールと統合するとともに、認証アーキテクチャが複雑化する中でベースラインを維持する必要があります。複数のアプリケーション用に 1 つの開始ポイントを作成する場合、ユーザーを正しいリソースにルーティングする最初の決定をキャプチャするユーザーインターフェイス (UI) 要素を構築する必要があります。

アプリケーションクライアントマルチテナンシーのベストプラクティス

アプリケーション内のテナントごとに [アプリケーションクライアント](#) を 1 つ作成します。アプリケーションクライアントマルチテナンシーを使用すると、テナントにリンクされたアプリケーションクライアントに任意のユーザーを割り当て、単一のユーザープロフィールを保持できます。ユーザープール内の [ID プロバイダー \(IdP\)](#) の一部またはすべてをアプリケーションクライアントに割り当てることができるため、テナントアプリケーションクライアントは、テナント固有の IdP によるサインインを許可できます。ユーザーが複数のテナントに存在する場合、プロフィールを複数の IdP にリンクすることで、ユーザーエクスペリエンスに一貫性を与えられます。

次の図は、共有ユーザープールに専用アプリケーションクライアントを持つ各テナントを示しています。



アプリケーションクライアントマルチテナンシーを実装するタイミング

Lambda トリガー、パスワードポリシー、E メールや SMS メッセージのコンテンツと配信方法など、ユーザープールレベルで設定のユニバーサル設定を選択できるとき。共有ユーザープールのユー

ザーは任意のアプリケーションクライアントにサインインできるため、アプリケーションクライアント固有の IdP や Amazon Cognito ユーザープール API によるサインインには、アプリケーションクライアントマルチテナンシーが最適です。アプリケーションクライアントマルチテナンシーは、複数のアプリケーション間での移行をユーザーに許可する場合の 1 対多環境にも適しています。

労力レベル

アプリケーションクライアントマルチテナンシーには、中程度の労力が必要になります。アプリケーションクライアントマルチテナンシーの主な課題は、テナントがホストされた UI Cookie を提示し、アプリケーションを切り替える機能です。アプリケーションクライアントマルチテナンシーアーキテクチャでは、隔離が必要となるホストされた UI サインインは避けてください。アプリケーションクライアントロジックを組み込んだモバイルアプリケーションやウェブアプリケーションへのリンクを配布することも、ユーザーのテナンシーを決定する初期 UI 要素を構築することもできます。複数のユーザープールとアイデンティティプールにおいて設定を標準化して維持する必要がないため、労力は少なくなります。

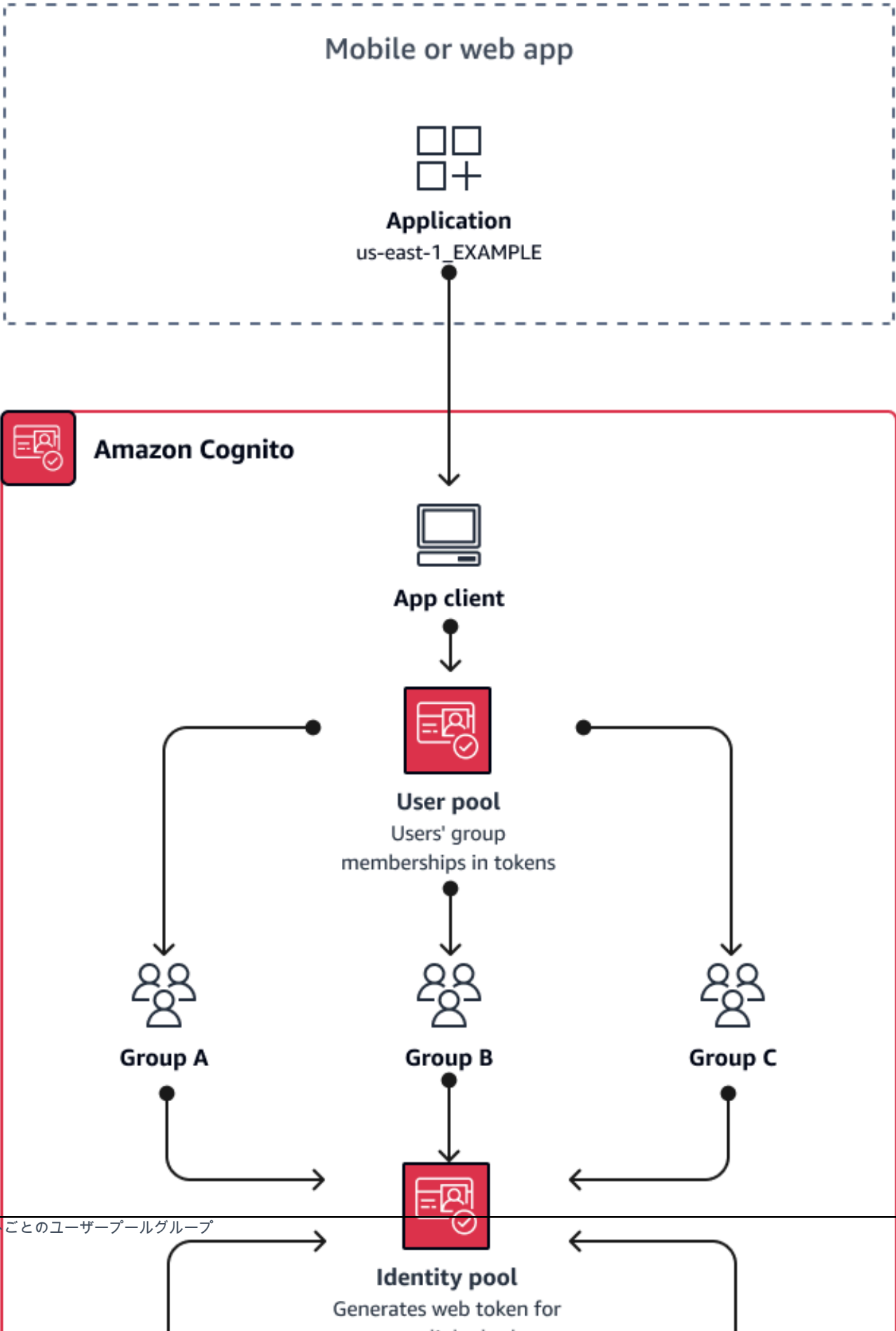
ユーザーグループのマルチテナンシーのベストプラクティス

グループベースのマルチテナンシーは、アーキテクチャでアイデンティティプールを持つ Amazon Cognito ユーザープールが必要な場合に最適です。

ユーザープールの [ID トークンとアクセストークン](#) には `cognito:groups` クレームが含まれています。さらに、ID トークンには `cognito:roles` クレームと `cognito:preferred_role` クレームが含まれます。アプリケーションでの認証の主な結果がアイデンティティプールからの一時的な AWS 認証情報である場合、ユーザーのグループメンバーシップによって、そのユーザーが受け取る [IAM ロール](#) とアクセス許可が決まります。

例えば、それぞれがアプリケーションアセットを独自の Amazon S3 バケットに保存している 3 つのテナントを考えてみましょう。各テナントのユーザーを、関連付けられたグループに割り当て、グループの優先ロールを設定し、そのロールにバケットへの読み取りアクセスを許可します。

次の図は、アプリケーションクライアントとユーザープールを共有するテナントで、IAM ロールの適格性を決定するユーザープール内の専用グループがあるものを示しています。



グループマルチテナンシーを実装するタイミング

AWS リソースへのアクセスが主な懸念事項である場合。Amazon Cognito ユーザープールのグループは、ロールベースのアクセス制御 (RBAC) のメカニズムです。ユーザープールに多数のグループを設定し、グループ優先度に基づく複雑な RBAC 決定を行うことができます。アイデンティティプールは、優先度が最も高いロール、グループクレーム内の任意のロール、またはユーザーのトークン内の他のクレームからのロールに対して、認証情報を割り当てることができます。

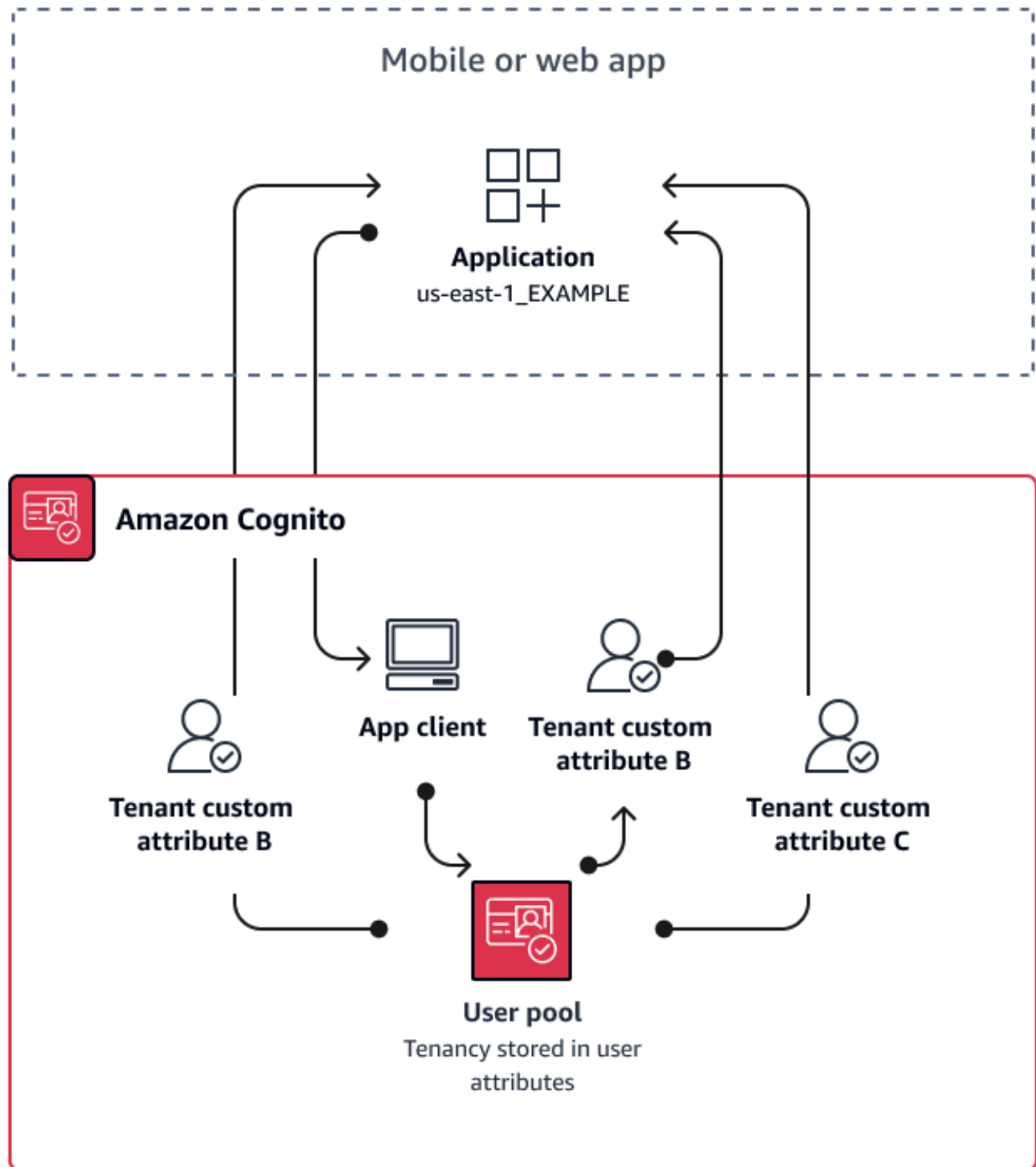
労力レベル

グループメンバーシップのみでマルチテナンシーを維持する労力は、低レベルです。ただし、ユーザープールグループのロールを IAM ロール選択の組み込み容量を超えて拡張するには、ユーザーのトークンにおけるグループメンバーシップを処理するアプリケーションロジックを構築し、クライアントで何をするかを決定する必要があります。Amazon Verified Permissions をアプリケーションと統合して、クライアント側の認可に関する決定を行うことができます。現在のところ、グループ識別子は Verified Permissions [IsAuthorizedWithToken](#) API オペレーションでは処理されませんが、グループメンバーシップクレームなど、トークンの内容を解析する[カスタムコードを開発](#)できます。

カスタム属性マルチテナンシーのベストプラクティス

Amazon Cognito は、選択した名前を持つ[カスタム属性](#)をサポートします。カスタム属性が役立つシナリオの 1 つは、共有ユーザープール内のユーザーのテナンシーを区別する場合です。custom:tenantID などの属性に対する値をユーザーに割り当てると、アプリケーションはそれに応じてテナント固有のリソースへのアクセスを割り当てることができます。テナント ID を定義するカスタム属性は、アプリケーションクライアントに対してイミュータブルまたは読み取り専用である必要があります。

次の図は、アプリケーションクライアントとユーザープールを共有するテナントで、そのテナントが属するユーザープール内のカスタム属性を持っているものを示しています。



カスタム属性がテナンシーを決定する場合、単一のアプリケーションまたはサインイン URL を配布できます。ユーザーがサインインすると、アプリケーションは、ロードするアセット、適用するブラ

ンド、表示する機能を決定する `custom:tenantID` クレームを処理できます。ユーザー属性からの高度なアクセスコントロールの決定については、Amazon Verified Permissions で ID プロバイダーとしてユーザープールを設定し、ID トークンやアクセストークンの内容からアクセス決定を生成します。

カスタム属性マルチテナンシーを実装するタイミング

テナンシーが表面レベルのとき。テナント属性は、ブランディングとレイアウトの結果に影響を与える可能性があります。テナント間での大幅な分離を実施する場合、カスタム属性は最適な選択ではありません。MFA やホストされた UI ブランディングなど、ユーザープールまたはアプリケーションクライアントのレベルで設定する必要がある形でテナント間に違いがある場合は、カスタム属性が提供しない方法でテナント間の区別を作成する必要があります。アイデンティティプールを使用すると、ID トークンのカスタム属性クレームから、ユーザーからの IAM ロールを選択することもできます。

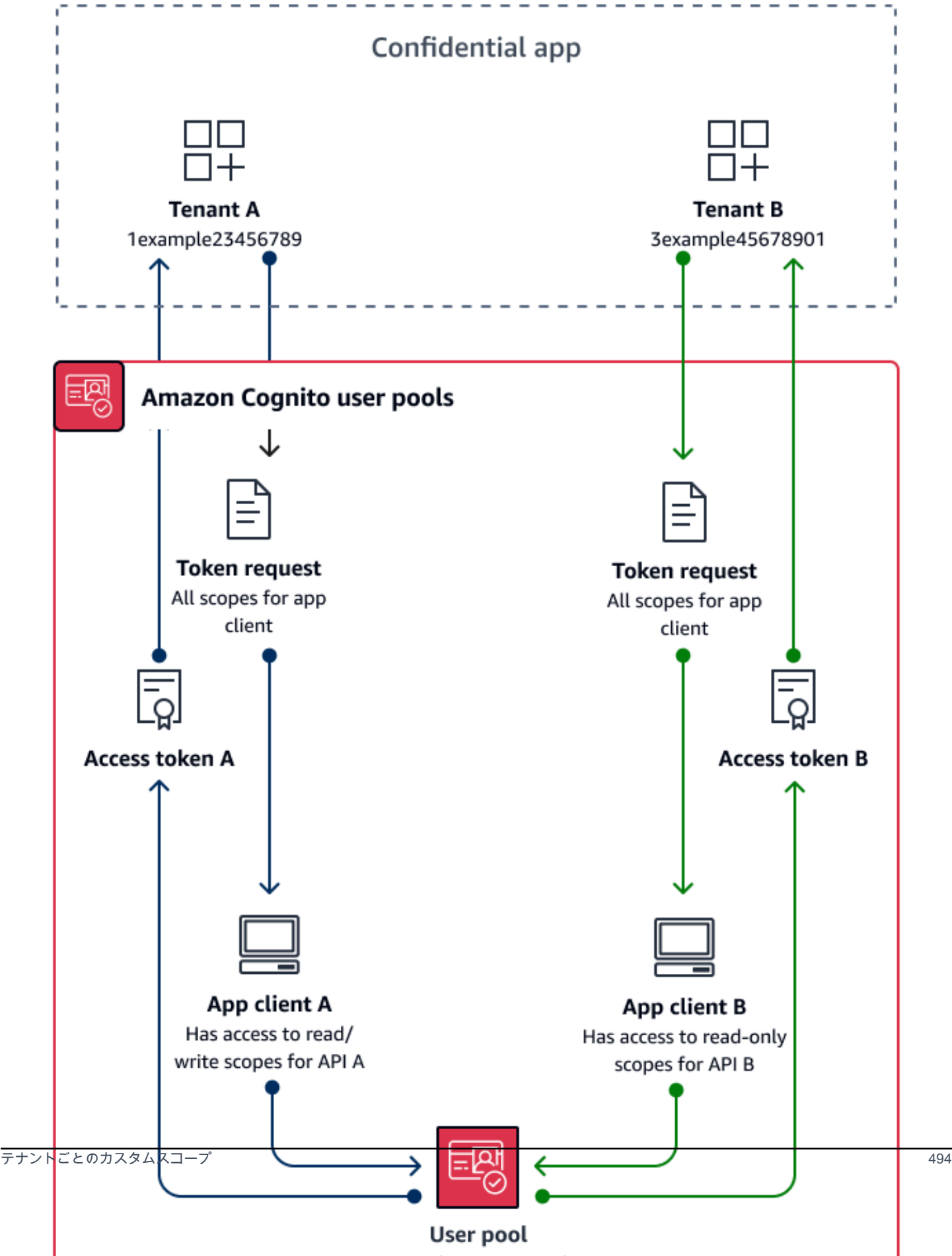
労力レベル

カスタム属性のマルチテナンシーでは、アプリケーションに対するテナントベースの認可に関する決定の義務が移転されるため、労力が高くなる傾向があります。OIDC クレームを解析するクライアント設定や、Amazon Verified Permissions に既に精通している場合、このアプローチに必要な労力は、最低レベルになる場合があります。

カスタムスコープマルチテナンシーのベストプラクティス

Amazon Cognito は、[リソースサーバー](#)用のカスタム OAuth 2.0 スコープをサポートしています。カスタムスコープを使用して、Machine to Machine (M2M) 認可モデル用のユーザープールにアプリケーションクライアントのマルチテナンシーを実装できます。スコープベースのマルチテナンシーでは、アプリケーションクライアントまたはアプリケーション設定でアクセスを定義することで M2M マルチテナンシーの実装に必要な労力が削減されます。

次の図は、カスタムスコープのマルチテナンシーのオプションを 1 つ示しています。ユーザープール内の関連スコープにアクセスできる専用アプリケーションクライアントを持つ各テナントを示しています。



カスタムスコープのマルチテナンシーを実装するタイミング

使用について、機密クライアントのクライアント認証情報による M2M 認証がなされるとき。ベストプラクティスとして、アプリケーションクライアント専用のリソースサーバーを作成します。カスタムスコープのマルチテナンシーは、リクエスト依存またはクライアント依存にすることができます。

リクエスト依存

テナントの要件に一致するスコープのみをリクエストするアプリケーションロジックを実装します。例えば、アプリケーションクライアントで API A と API B への読み取りと書き込みのアクセスを発行できる場合であっても、テナントアプリケーション A が API A の読み取りスコープとテナンシーを示すスコープのみをリクエストするというものです。このモデルにより、テナント間の共有スコープについて、複雑性の高い組み合わせが可能になります。

クライアント依存

認可リクエストでアプリケーションクライアントに割り当てられたすべてのスコープをリクエストします。これを行うには、[トークンエンドポイント](#) へのリクエストにおける scope リクエストパラメータを除外します。このモデルにより、アプリケーションクライアントは、カスタムスコープに追加するアクセスインジケータを保存できます。

いずれの場合も、アプリケーションは、依存するデータソースの権限を示すスコープを持つアクセストークンを受け取ります。また、スコープは、アプリケーションに他の情報を提示することもできます。

- テナンシーを指定する
- リクエストログに役立つ
- アプリケーションがクエリを許可されている API を示す
- アクティブな顧客の初期チェックに情報を提供する

労力レベル

カスタムスコープのマルチテナンシーには、アプリケーションの規模に応じてさまざまなレベルの労力が必要になります。アプリケーションがアクセストークンを解析し、適切な API リクエストを実行できるようにするアプリケーションロジックを考案する必要があります。

例えば、リソースサーバスコープは [resource server identifier]/[name] の形式で指定されます。リソースサーバー識別子は、テナントスコープからの認可に関する決定に関連している可能性が低く、スコープ名を一貫して解析する必要があります。

リソースの例

次の AWS CloudFormation テンプレートは、1 つのリソースサーバーとアプリケーションクライアントを使用して、カスタムスコープのマルチテナンシー用のユーザープールを作成します。

```
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"
```

マルチテナンシーのセキュリティに関する推奨事項

アプリケーションの安全性を高めるために、以下を推奨します。

- Amazon Verified Permissions を使用して、アプリケーションのテナンシーを検証します。アプリケーションでユーザーのリクエストを許可する前に、ユーザープール、アプリケーションクライアント、グループ、またはカスタム属性の使用権限を調べるポリシーを構築します。Amazon Cognito ユーザープールを念頭に置いて Verified Permissions [ID ソース](#) AWS を作成しました。Verified Permissions には、マルチテナンシー管理に関する[追加のガイダンス](#)があります。
- ドメインの一致に基づいたテナントへのユーザーアクセスを承認するために、検証済みの E メールアドレスのみを使用します。E メールアドレスと電話番号は、アプリが検証するか、外部 IdP が検証の証明を提供しない限り、信頼しないでください。これらの許可を設定する方法の詳細については、「[属性の許可と範囲](#)」を参照してください。
- テナントを識別するユーザープロフィール属性には、イミュータブルカスタム属性または読み取り専用カスタム属性を使用します。ユーザーを作成するか、ユーザープールにサインアップする場合にのみ、イミュータブル属性の値を設定できます。また、アプリケーションに属性への読み取り専用アクセスを許可します。
- 外部 IdP とアプリケーションクライアントの間の 1 対 1 マッピングを使用して、認可されていないクロステナントアクセスを防止します。外部 IdP によって認証され、有効な Amazon Cognito セッション Cookie を持つユーザーは、同じ IdP を信頼する他のテナントアプリケーションにアクセスできます。
- アプリケーションにテナントマッチングおよび認可ロジックを実装する場合は、テナントへのユーザーアクセスを認可する基準を変更できないようにユーザーを制限してください。また、フェデ

レーションのために外部 IdP が使用されている場合は、テナント ID プロバイダー管理者がユーザーアクセスを変更できないように制限してください。

一般的な Amazon Cognito シナリオ

このトピックでは、Amazon Cognito を使用するための 6 つの一般的なシナリオについて説明します。

Amazon Cognito の主な 2 つのコンポーネントは、ユーザープールと ID プールです。ユーザープールは、ウェブおよびモバイルユーザーにサインアップとサインインオプションを提供するユーザーディレクトリです。ID プールは、ユーザーに他の へのアクセスを許可するための一時的な AWS 認証情報を提供します AWS のサービス。

ユーザープールは、Amazon Cognito のユーザーディレクトリです。アプリケーションユーザーは、ユーザープール経由で直接サインインするか、またはサードパーティーの ID プロバイダー (IdP) 経由でフェデレーション方式で認証を行うことができます。ユーザープールは、Facebook、Google、Amazon、Apple 経由のソーシャルサインイン、および OpenID Connect (OIDC) と SAML IdP から返されるトークンの処理のオーバーヘッドを管理します。ユーザーが直接またはサードパーティーを通じてサインインするかどうかにかかわらず、ユーザープールのすべてのメンバーには、SDK を通じてアクセスできるディレクトリプロファイルがあります。

ID プールを使用すると、ユーザーは Amazon S3 や DynamoDB などのサービスにアクセス AWS するための一時的な AWS 認証情報を取得できます。ID プールは、匿名ゲストユーザーに加えて、サードパーティー IdP 経由のフェデレーションもサポートしています。

トピック

- [ユーザープールを使用して認証する](#)
- [ユーザープールトークンを使用するバックエンドリソースへのアクセス](#)
- [ユーザープールと共に API Gateway と Lambda を使用してリソースにアクセスする](#)
- [ユーザープールと ID プールを使用して AWS サービスにアクセスする](#)
- [サードパーティーによる認証と ID プールによる AWS サービスへのアクセス](#)
- [Amazon Cognito で AWS AppSync リソースにアクセスする](#)

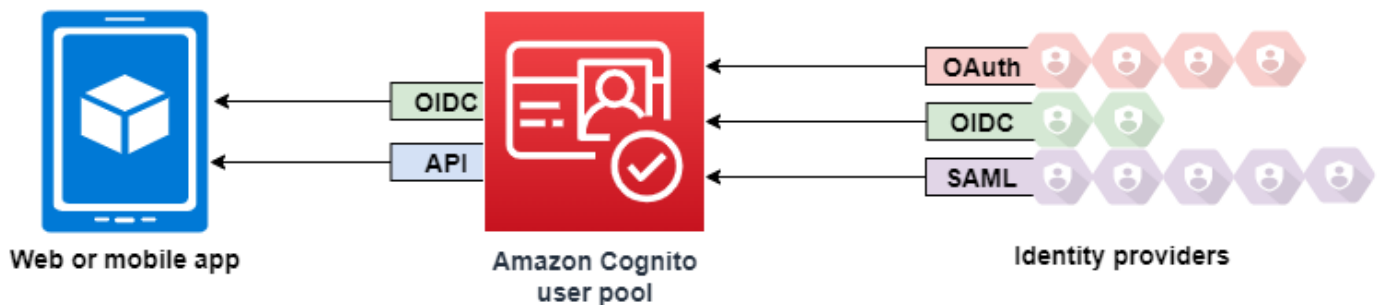
ユーザープールを使用して認証する

ユーザーがユーザープールを使用して認証できるようにすることが可能です。アプリケーションユーザーは、ユーザープール経由で直接サインインするか、またはサードパーティーの ID プロバイダー (IdP) 経由でフェデレーション方式で認証を行うことができます。ユーザープール

は、Facebook、Google、Amazon、Apple 経由のソーシャルサインイン、および OpenID Connect (OIDC) と SAML IdP から返されるトークンの処理のオーバーヘッドを管理します。

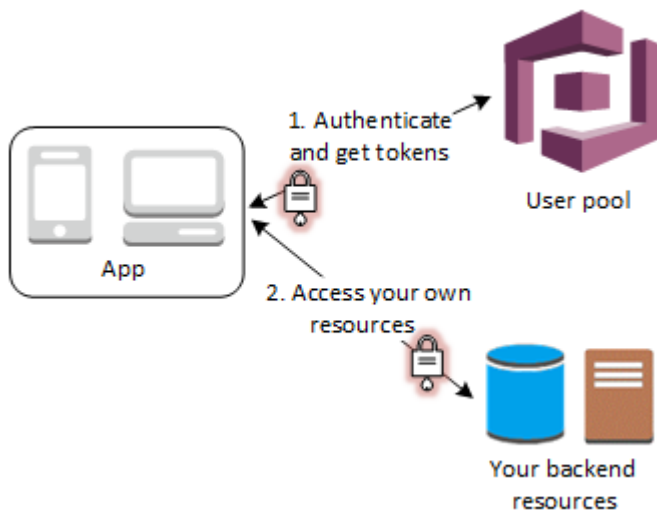
認証が正常に行われると、ウェブまたはモバイルアプリが Amazon Cognito からユーザープールトークンを受け取ります。これらのトークンを使用して、アプリが他の AWS サービスにアクセスできるようにする AWS 認証情報を取得できます。または、それらを使用してサーバー側のリソースまたは Amazon API Gateway へのアクセスを制御することもできます。

詳細については、「[認証セッションの例](#)」および「[ユーザープール JSON ウェブトークン \(JWT\) の理解](#)」を参照してください。



ユーザープールトークンを使用するバックエンドリソースへのアクセス

ユーザープールへのサインインが正常に行われると、ウェブまたはモバイルアプリが Amazon Cognito からユーザープールトークンを受け取ります。サーバー側のリソースへのアクセスを制御するには、これらのトークンを使用します。また、ユーザープールグループを作成して許可を管理したり、異なるタイプのユーザーを表したりすることもできます。グループを使用して、リソースをアクセス制御する方法の詳細については、「[ユーザープールにグループを追加する](#)」を参照してください。



ユーザープールのドメインを設定した後、Amazon Cognito が、アプリにサインアップおよびサインインページを追加できるようにするホストされたウェブの UI をプロビジョニングします。この OAuth 2.0 認証基盤を使用することで、独自のリソースサーバーを作成でき、ユーザーは保護されたリソースにアクセスできるようになります。詳細については、「[スコープ、M2M、リソースサーバー](#)」を参照してください。

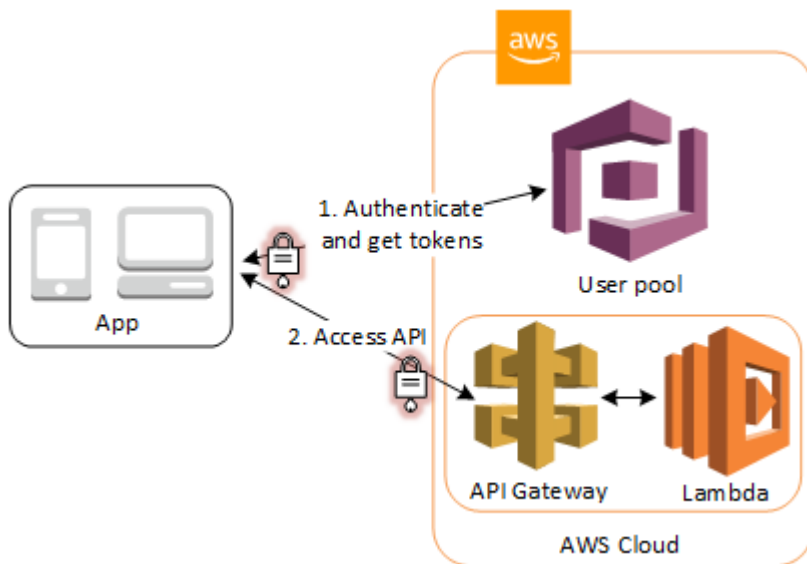
ユーザープール認証の詳細については、「[認証セッションの例](#)」および「[ユーザープール JSON ウェブトークン \(JWT\) の理解](#)」を参照してください。

ユーザープールと共に API Gateway と Lambda を使用してリソースにアクセスする

ユーザーが API Gateway 経由で API にアクセスできるようにすることが可能です。API Gateway は、正常に行われたユーザープール認証からのトークンを検証し、これらのトークンを Lambda 関数などのリソース、または独自の API へのアクセス権をユーザーに付与するために使用します。

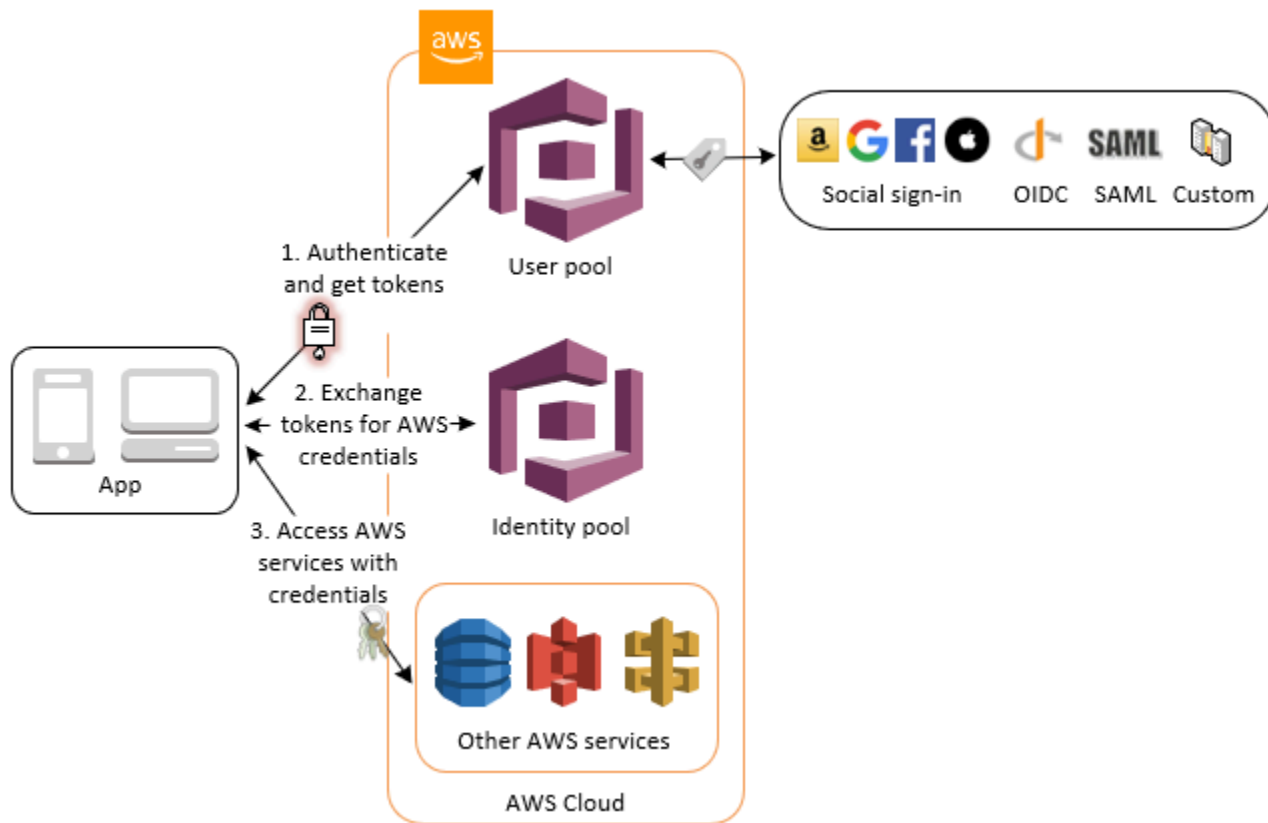
ユーザープール内のグループは、グループメンバーシップを IAM ロールにマップすることによって、API Gateway でアクセス許可を制御するために使用できます。ユーザーがメンバーとなっているグループは、アプリユーザーのサインイン時にユーザープールより付与される ID トークンに含まれます。ユーザープールグループの詳細については、「[ユーザープールにグループを追加する](#)」を参照してください。

Amazon Cognito オーソライザーの Lambda 関数による検証のために、API Gateway へのリクエストでユーザープールトークンを送信することができます。API Gateway の詳細については、「[Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する](#)」を参照してください。



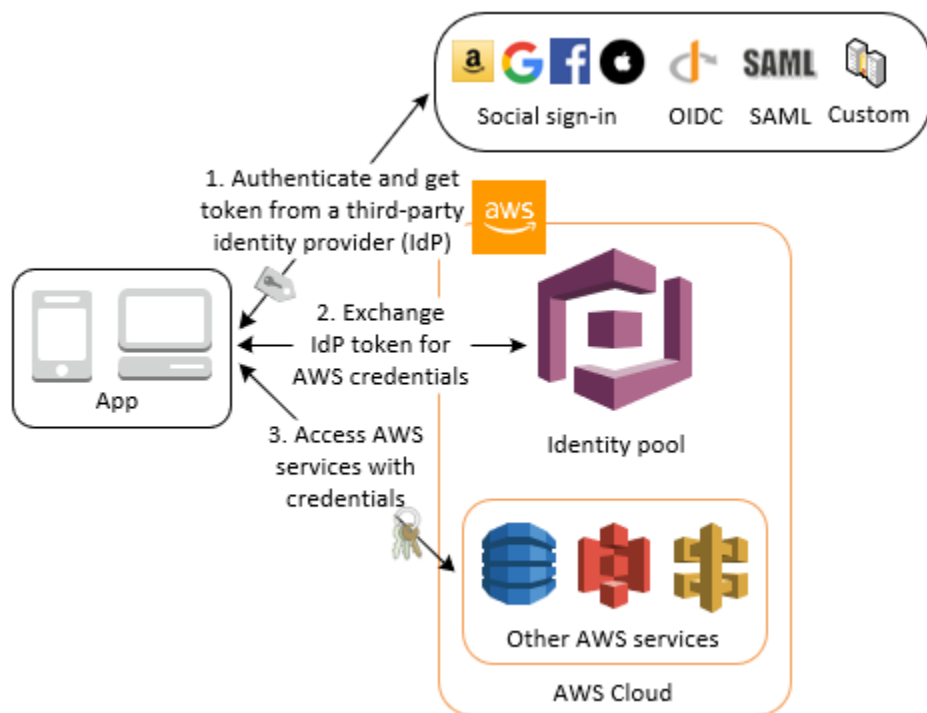
ユーザープールと ID プールを使用して AWS サービスにアクセスする

ユーザープールの認証が正常に行われると、アプリケーションが Amazon Cognito からユーザープールトークンを受け取ります。ID プールを使用して、他の AWS サービスへの一時的なアクセスと交換できます。詳細については、「[サインイン後の ID プール AWS のサービスを使用した へのアクセス](#)」および「[Amazon Cognito アイデンティティプールの開始方法](#)」を参照してください。



サードパーティーによる認証と ID プールによる AWS サービスへのアクセス

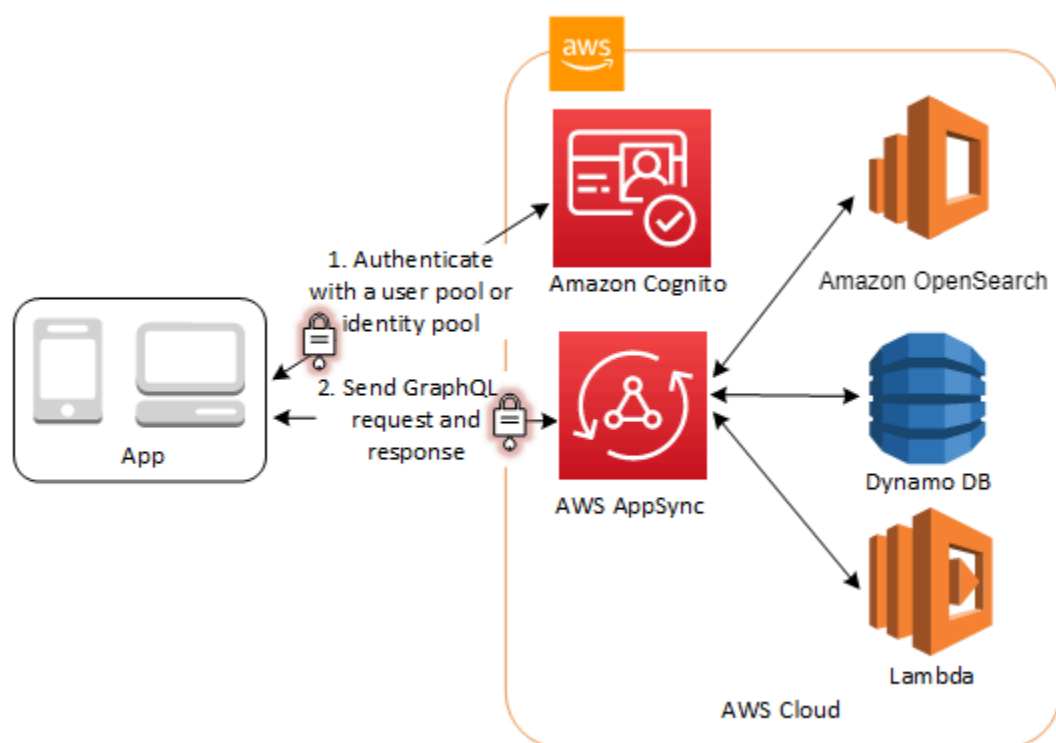
ID プールを介して AWS のサービスへのアクセスをユーザーに許可できます。ID プールには、サードパーティー ID プロバイダーによって認証されたユーザーからの IdP トークンが必要です (匿名ゲストの場合は何も必要ありません)。代わりに、ID プールは、他の AWS サービスにアクセスするために使用できる一時的な AWS 認証情報を付与します。詳細については、「[Amazon Cognito アイデンティティプールの開始方法](#)」を参照してください。



Amazon Cognito で AWS AppSync リソースにアクセスする

成功した Amazon Cognito ユーザープール認証からのトークンを使用して、AWS AppSync リソースへのアクセス権をユーザーに付与できます。詳細については、「AWS AppSync デベロッパーガイド」の「[AMAZON_COGNITO_USER_POOLS authorization](#)」を参照してください。

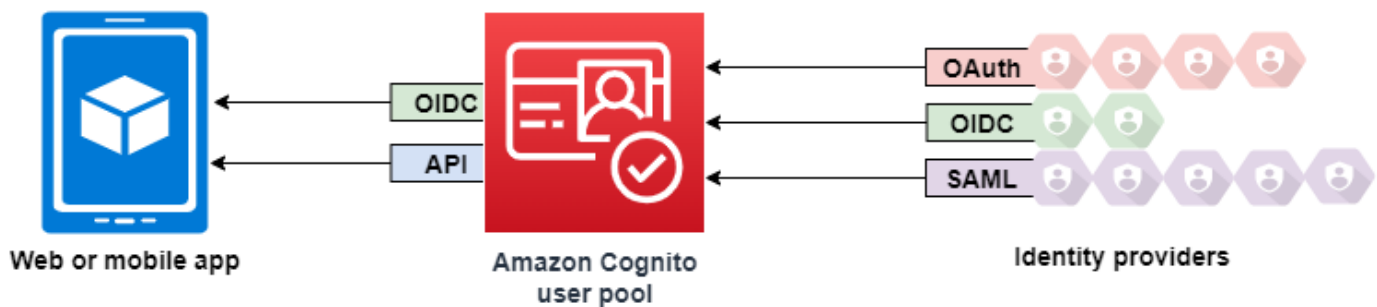
ID プールから受け取った IAM 認証情報を使用して、AWS AppSync GraphQL API へのリクエストに署名することもできます。「[AWS IAM authorization](#)」を参照してください。



Amazon Cognito user pools

Amazon Cognito ユーザープールは、ウェブおよびモバイルアプリケーションの認証と認可のためのユーザーディレクトリです。アプリケーションの観点から見ると、Amazon Cognito ユーザープールは OpenID Connect (OIDC) ID プロバイダー (OIDC) です。ユーザープールには、セキュリティ、ID フェデレーション、アプリ統合、ユーザーエクスペリエンスのカスタマイズなどの機能が何層も追加されます。

例えば、ユーザーのセッションが信頼できるソースからのものであることを確認できます。Amazon Cognito ディレクトリを外部 ID プロバイダーと組み合わせることができます。お好みの AWS SDK を使用して、アプリに最適な API 認可モデルを選択できます。また、Amazon Cognito のデフォルトの動作を変更またはオーバーホールする AWS Lambda 関数を追加することもできます。



トピック

- [機能](#)
- [ユーザープールの機能プラン](#)
- [Amazon Cognito ユーザープールに対するセキュリティのベストプラクティス](#)
- [Amazon Cognito ユーザープールによる認証](#)
- [サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン](#)
- [ユーザープールのマネージドログイン](#)
- [Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#)
- [ユーザープール内のユーザーを管理する](#)
- [ユーザープール JSON ウェブトークン \(JWT\) の理解](#)
- [サインイン完了後にリソースにアクセスする](#)
- [スコープ、M2M、リソースサーバー](#)
- [ユーザープールの機能を設定する](#)

- [Amazon Cognito ユーザープールのセキュリティ機能を使用する](#)
- [ユーザープールのエンドポイントとマネージドログインのリファレンス](#)

機能

Amazon Cognito ユーザープールには次の機能があります。

サインアップ

Amazon Cognito ユーザープールには、ユーザープールにユーザープロフィールを追加するためのユーザー主導型、管理者主導型、およびプログラムによる方法があります。Amazon Cognito ユーザープールは、以下のサインアップモデルをサポートしています。アプリでは、これらのモデルを任意に組み合わせて、アプリで使用できます。

Important

ユーザープールでユーザーサインアップを有効にすると、インターネット上のすべてのユーザーがアカウントにサインアップしてアプリケーションにサインインできるようになります。アプリケーションをパブリックサインアップで開く場合を除き、ユーザープールで自己登録を有効にしないでください。この設定を変更するには、ユーザープールコンソールの[認証]の下にある[サインアップ]メニューで[セルフサービスのサインアップ]を更新するか、[CreateUserPool](#) または [UpdateUserPool](#) API リクエストで [AllowAdminCreateUserOnly](#) の値を更新します。

ユーザープールに設定できるセキュリティ機能については、「[Amazon Cognito ユーザープールのセキュリティ機能を使用する](#)」を参照してください。

1. ユーザーはアプリに情報を入力し、ユーザープールにネイティブなユーザープロフィールを作成できます。API サインアップオペレーションを呼び出して、ユーザープールにユーザーを登録できます。これらのサインアップ操作は誰にでも公開することも、クライアントシークレットまたは AWS 認証情報を使用して認可することもできます。
2. ユーザーを Amazon Cognito への情報の受け渡しを許可できるサードパーティーの IdP にリダイレクトできます。Amazon Cognito は、OIDC ID トークン、OAuth 2.0 userInfo データ、および SAML 2.0 アサーションを処理して、ユーザープールのユーザープロフィールに入れます。Amazon Cognito に受信させる属性は、属性マッピングルールに基づいて制御します。
3. パブリックサインアップやフェデレーションサインアップをスキップして、独自のデータソースとスキーマに基づいてユーザーを作成できます。Amazon Cognito コンソールまたは API でユー

ザーを直接追加できます。CSV ファイルからのユーザーをインポートします。既存のディレクトリで新しいユーザーを検索し、既存のデータからユーザープロフィールを入力する just-in-time AWS Lambda 関数を実行します。

ユーザーがサインアップしたら、Amazon Cognito がアクセストークンと ID トークンにリストするグループにユーザーを追加できます。ID トークンをアイデンティティプールに渡すときに、ユーザープールグループを IAM ロールにリンクすることもできます。

関連トピック

- [ユーザープール内のユーザーを管理する](#)
- [API、OIDC、マネージドログインページの認証についての理解](#)
- [SDK を使用した Amazon Cognito ID プロバイダーのコード例 AWS SDKs](#)

サインイン

Amazon Cognito は、スタンドアロンのユーザーディレクトリおよび ID プロバイダー (IdP) として使用し、アプリケーションの ID プロバイダー (IdP) として使用できます。ユーザーは、Amazon Cognito がホストするマネージドログインページを使用するか、Amazon Cognito ユーザープール API を介してカスタム構築されたユーザー認証サービスを使用してサインインできます。カスタム構築されたフロントエンドの背後にあるアプリケーション層は、正当なリクエストであることを確認する複数の方法のいずれかを使用して、リクエストをバックエンドで承認できます。

ユーザーは、ユーザー名とパスワード、パスキー、E メールおよび SMS メッセージのワンタイムパスワードを設定してサインインできます。外部ユーザーディレクトリとの統合サインイン、サインイン後の多要素認証 (MFA)、記憶されているデバイスの信頼、独自に設計したカスタム認証フローを提供できます。

外部ディレクトリ (オプションで Amazon Cognito に組み込まれたユーザーディレクトリと組み合わせる) を使用してユーザーをサインインさせるには、次の統合を追加できます。

1. OAuth 2.0 ソーシャルサインインを使用してサインインし、顧客ユーザーデータをインポートします。Amazon Cognito は、OAuth 2.0 による Google、Facebook、Amazon、および Apple へのサインインをサポートしています。
2. SAML および OIDC サインインを使用してサインインし、職場および学校のユーザーデータをインポートします。Amazon Cognito は、任意の SAMID Connect (OIDC) ID プロバイダー (IdP) からのクレームを受け入れるように設定することもできます。

3. 外部ユーザープロファイルをネイティブユーザープロファイルにリンクします。リンクされたユーザーは、サードパーティーのユーザー ID を使用してサインインし、組み込みディレクトリのユーザーに割り当てたアクセス権を受け取ることができます。

関連トピック

- [サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン](#)
- [フェデレーションユーザーを既存のユーザープロファイルにリンクする](#)

マシンツーマシン認可

一部のセッションは、人間からマシンへのインタラクションではありません。自動プロセスによって API へのリクエストを承認できるサービスアカウントが必要になる場合があります。OAuth 2.0 スコープを使用して Machine to Machine 認証用のアクセストークンを生成するには、[クライアント認証情報の付与](#)を生成するアプリケーションクライアントを追加します。

関連トピック

- [スコープ、M2M、リソースサーバー](#)

マネージドログイン

ユーザーインターフェイスを構築しない場合は、カスタマイズしたマネージドログインページをユーザーに提供できます。マネージドログインとは、サインアップ、サインイン、多要素認証 (MFA)、パスワードリセットのための一連のウェブページです。マネージドログインを既存のドメインに追加するか、AWS サブドメインでプレフィックス識別子を使用できます。

関連トピック

- [ユーザープールのマネージドログイン](#)
- [ユーザープールのドメインを設定する](#)

セキュリティ

ローカルユーザーは、SMS メッセージや Eメールメッセージのコード、または多要素認証 (MFA) コードを生成するアプリケーションを使用して、追加の認証要素を提供できます。アプリケーション内で MFA を設定および処理するメカニズムを構築することも、マネージドログインで MFA を管理

することも可能です。Amazon Cognito ユーザープールは、ユーザーが信頼できるデバイスからサインインするときに MFA をバイパスできます。

最初にユーザーに MFA を要求しない場合は、条件付きで要求できます。アダプティブ認証を使用すると、Amazon Cognito は悪意のある可能性があるアクティビティを検出し、ユーザーに MFA の設定を要求したり、サインインをブロックしたりすることができます。

ユーザープールへのネットワークトラフィックが悪意のある可能性がある場合は、モニタリングして AWS WAF ウェブ ACLs でアクションを実行できます。

関連トピック

- [ユーザープールに MFA を追加します](#)
- [脅威保護を備えた高度なセキュリティ](#)
- [AWS WAF ウェブ ACL をユーザープールに関連付ける](#)

カスタマーユーザーエクスペリエンス

ユーザーのサインアップ、サインイン、またはプロフィール更新のほとんどの段階で、Amazon Cognito がリクエストを処理する方法をカスタマイズできます。Lambda トリガーを使用すると、ID トークンを変更したり、カスタム条件に基づいてサインアップリクエストを拒否したりできます。自分のカスタム認証フローを作成できます。

カスタム CSS とロゴをアップロードして、マネージドログインの外観と操作感をユーザーが使い慣れたものにすることができます。

関連トピック

- [Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#)
- [カスタム認証チャレンジの Lambda トリガー](#)
- [マネージドログインページにブランディングを適用する](#)

モニタリングと分析

Amazon Cognito ユーザープールは、マネージドログインへのリクエストなどの API リクエストを AWS CloudTrail に記録します。パフォーマンスメトリクスを Amazon CloudWatch Logs で確認し、Lambda トリガーを使用してカスタムログを CloudWatch にプッシュできます。また、E メール

メッセージと SMS メッセージの配信状況をモニタリングし、Service Quotas コンソールで API リクエストの量をモニタリングできます。

プラス機能プランでは、自動学習テクノロジーでユーザーの認証試行をモニタリングして漏えいの兆候がないか確認し、リスクを即座に修復できます。また、これらの高度なセキュリティ機能では、ユーザーアクティビティをユーザープールに記録します。必要に応じて Amazon S3、CloudWatch Logs、または Amazon Data Firehose にも記録します。

API リクエストのデバイスおよびセッションデータを Amazon Pinpoint キャンペーンに記録することもできます。Amazon Pinpoint では、ユーザーアクティビティの分析に基づいて、アプリからプッシュ通知を送信できます。

関連トピック

- [Amazon Cognito ログイン AWS CloudTrail](#)
- [CloudWatch と Service Quotas でのクォータと使用状況の追跡](#)
- [Amazon Cognito ユーザープールからのログのエクスポート](#)
- [ユーザープール分析に Amazon Pinpoint を使用する](#)

Amazon Cognito アイデンティティプール統合

Amazon Cognito のもう半分はアイデンティティプールです。アイデンティティプールは、ユーザーからの Amazon DynamoDB や Amazon S3 など、AWS のサービスに対して API リクエストを認可およびモニタリングするための認証情報を提供します。ユーザープール内のユーザーをどのように分類したかに基づいてデータを保護する ID ベースのアクセスポリシーを構築できます。アイデンティティプールは、ユーザープールの認証とは無関係に、さまざまな ID プロバイダーからのトークンや SAML 2.0 アサーションを受け入れることもできます。

関連トピック

- [サインイン後の ID プール AWS のサービスを使用した へのアクセス](#)
- [Amazon Cognito アイデンティティプール](#)

ユーザープールの機能プラン

コストを理解することは、Amazon Cognito ユーザープール認証の実装準備における重要なステップです。Amazon Cognito には、ユーザープール向けの機能プランがあります。各プランには、一連の

機能とアクティブなユーザーあたりの月額コストが設定されています。機能プランの階層が上がるほど、それより下の機能プランより多くの機能を利用できます。

ユーザープールには、オンとオフを切り替えることができるさまざまな機能があります。例えば、多要素認証 (MFA) をオンにし、サードパーティ ID プロバイダー (IdP) によるサインインをオフにすることができます。一部の変更を行うには、機能プランを切り替える必要があります。ユーザープールの次の特性によって、毎月 AWS 請求する使用コストが決まります。

- 選択した機能
- アプリケーションがユーザープール API に対して行う 1 秒あたりのリクエスト数
- 1 か月に認証、更新、またはクエリアクティビティを行ったユーザーの数。[月間アクティブユーザー](#) (MAU) 数とも呼ばれます。
- サードパーティの SAML 2.0 または OpenID Connect (OIDC) IdP からの月間アクティブユーザー数
- Machine to Machine 認証のためにクライアント認証情報の付与を行うアプリケーションクライアントとユーザープールの数

ユーザープールの料金に関する最新情報については、「[Amazon Cognito の料金](#)」を参照してください。

機能プランの選択は、1 つのユーザープールに適用されます。同じ AWS アカウント 内の異なるユーザープールごとに、異なるプランを選択できます。ユーザープール内のアプリケーションクライアントに個別の機能プランを適用することはできません。新しいユーザープールにデフォルトで選択されるプランは、エッセンシャルです。

アプリケーションの要件に合わせて、いつでも機能プランを切り替えることができます。プランの切り替えに伴う変更によっては、アクティブな機能をオフにする必要があります。詳細については、「[機能をオフにして機能プランを変更する](#)」を参照してください。

ユーザープールの機能プラン

ライト

ライトは、月間アクティブユーザー数が少ないユーザープール向けの低コストの機能プランです。ユーザーディレクトリの認証機能が基本的なものである場合は、このプランで十分です。このプランには、サインイン機能と、クラシックのホストされた UI (マネージドログインの前身で、よりスリムでカスタマイズ性が低い) が含まれています。ライトプランには、アクセストークンのカスタマイズやパスキー認証など、多くの新しい機能は含まれていません。

エッセンシャル

エッセンシャルには、最新のユーザープール認証機能がすべて含まれています。このプランは、アプリケーションに新しいオプションを追加します。ログインページがマネージドログインであるか、カスタム構築であるかは問いません。エッセンシャルには、[\[選択ベースのサインイン\]](#)や[\[E メール MFA\]](#)などの高度な認証機能があります。

Plus

プラスには、エッセンシャルプランのすべての機能に加えて、ユーザーを保護するための高度なセキュリティ機能が含まれています。ユーザーのサインイン、サインアップ、パスワード管理のリクエストをモニタリングして、漏えいの兆候がないか確認します。例えば、ユーザープールでは、ユーザーが予期しない場所からサインインしていないか、公衆への情報漏洩につながるパスワードを使用していないかを検出できます。

プラスプランのユーザープールは、ユーザーアクティビティの詳細とリスク評価のログを生成します。これらのログを外部サービスにエクスポートするときに、独自の使用状況とセキュリティの分析を適用できます。

Note

以前は、一部のユーザープール機能が「高度なセキュリティ機能」料金体系に含まれていました。この体系に含まれていた機能は、現在、エッセンシャルプランまたはプラスプランに含まれています。

トピック

- [機能プランを選択する](#)
- [プラン別の機能](#)
- [エッセンシャルプランの機能](#)
- [プラスプランの機能](#)
- [機能をオフにして機能プランを変更する](#)

機能プランを選択する

AWS マネジメントコンソール

機能プランを選択するには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択、または[ユーザープールを作成](#)します。
4. [設定] メニューを選択し、[機能プラン] タブを確認します。
5. ライト、エッセンシャル、プラスの各プランで利用できる機能を確認します。
6. プランを変更するには、[エッセンシャルに切り替える] または [プラスに切り替える] を選択します。ライトプランに切り替えるには、[その他のプラン]、[ライトと比較] の順に選択します。
7. 次の画面で、選択内容を確認し、[確認] を選択します。

CLI/API/SDK

[CreateUserPool](#) オペレーションと [UpdateUserPool](#) オペレーションでは、UserPoolTier パラメータで機能プランを設定します。UserPoolTier の値を指定しない場合、ユーザープールのデフォルトは Essentials です。AdvancedSecurityMode を AUDIT または ENFORCED に設定する場合、ユーザープール階層は PLUS にする必要があり、指定しない場合はデフォルトで PLUS になります。

構文については、[CreateUserPool の「例」](#)を参照してください。さまざまなプログラミング言語の SDK のこの関数へのリンクについては、「[CreateUserPool の」も参照](#)してください。AWS SDKs

```
"UserPoolTier": "PLUS"
```

では AWS CLI、このオプションは `--user-pool-tier` 引数です。

```
--user-pool-tier PLUS
```

詳細については、AWS CLI コマンドリファレンスの [create-user-pool](#) と [update-user-pool](#) を参照してください。

プラン別の機能

ユーザープールの機能とプラン

機能	説明	機能プラン
安全でないパスワードからの保護	プレーンテキストのパスワードに漏えいの兆候がないかランタイムに確認する	Plus
悪意のあるログイン試行からの保護	セッションプロパティに侵害の兆候がないかランタイムに確認する	Plus
ユーザーアクティビティの記録と分析	ユーザー認証セッションのプロパティとリスクスコアのログを生成する	Plus
ユーザーアクティビティログのエクスポート	ユーザーセッションとリスクログを外部にプッシュする AWS のサービス	Plus
ビジュアルエディタによるマネージドログインページのカスタマイズ	Amazon Cognito コンソールのビジュアルエディタを使用して、マネージドログインページにブランディングとスタイルを適用する	エッセンシャル + プラス
メールワンタイムコード付きの MFA	ユーザー名の認証後に、追加の E メールメッセージサインイン要素の提供をローカルユーザーにリクエストまたは要求する	エッセンシャル + プラス
実行時にアクセストークンのスコープとクレームをカスタマイズ	Lambda トリガーを使用してユーザープールのアクセストークンの認証機能を拡張する	エッセンシャル + プラス

機能	説明	機能プラン
ワンタイムコードによるパスワードなしのサインイン	最初の認証要素として E メールまたは SMS でワンタイムパスワードを受け取ることをユーザーに許可する	エッセンシャル + プラス
ハードウェアまたはソフトウェアの FIDO2 Authenticator を使用したパスキーサインイン	FIDO2 Authenticator に保存されている暗号化キーを最初の認証要素として使用することをユーザーに許可する	エッセンシャル + プラス
サインアップとサインイン	認証オペレーションを実行し、新規ユーザーがアプリケーションのアカウントに登録できるようにする	ライト + エッセンシャル + プラス
ユーザーグループ	ユーザーの論理グループを作成し、アイデンティティプールのオペレーションにデフォルトの IAM ロールを割り当てる	ライト + エッセンシャル + プラス
ソーシャルプロバイダー、SAML プロバイダー、OIDC プロバイダーによるサインイン	直接サインインするか、希望するプロバイダーを使用してサインインするかのオプションをユーザーに提供する	ライト + エッセンシャル + プラス
OAuth 2.0/OIDC 認可サーバー	OIDC 発行者として機能する	ライト + エッセンシャル + プラス
ログインページ	ホストされた認証用ウェブページのコレクション。マネージドログインは、エッセンシャル階層とプラス階層で使用できます。クラシックのホストされた UI は、すべての機能階層で使用できます。	ライト + エッセンシャル + プラス

機能	説明	機能プラン
パスワード、カスタム、更新トークン、SRP 認証	アプリケーションでユーザー名とパスワードの入力をユーザーに求める	ライト + エッセンシャル + プラス
クライアント認証情報を使用した Machine to Machine (M2M)	人間以外のエンティティを認証するためのアクセストークンを発行する	ライト + エッセンシャル + プラス
リソースサーバーによる API 認証	外部システムへのアクセスを許可するカスタムスコープを使用してアクセストークンを発行する	ライト + エッセンシャル + プラス
ユーザーのインポート	CSV ファイルからのインポートジョブを設定し、ユーザーのサインイン時にジャストインタイム移行を実行する	ライト + エッセンシャル + プラス
Authenticator アプリケーションと SMS ワンタイムコードを使用した MFA	ユーザー名の認証後に追加の SMS メッセージまたは Authenticator アプリケーションのサインイン要素の提供をローカルユーザーにリクエストまたは要求する	ライト + エッセンシャル + プラス
ID トークンのスコープとクレームのランタイムカスタマイズ	Lambda トリガーを使用してユーザープールのアイデンティティ (ID) トークンの認証機能を拡張する	ライト + エッセンシャル + プラス
Lambda トリガーを使用したカスタムランタイムアクション	外部アクションを実行し、認証に影響を与える Lambda 関数を使用して、サインインプロセスをランタイムにカスタマイズする	ライト + エッセンシャル + プラス

機能	説明	機能プラン
CSS によるマネージドログインページのカスタマイズ	CSS テンプレートをダウンロードし、マネージドログインページでいくつかのスタイルを変更する	ライト + エッセンシャル + プラス

エッセンシャルプランの機能

エッセンシャル機能プランには、Amazon Cognito ユーザープールの最良かつ最新の機能のほとんどが含まれています。ライトプランからエッセンシャルプランに切り替えると、マネージドログインページの新機能、E メールメッセージのワンタイムパスワードを使用した多要素認証、拡張パスワードポリシー、カスタムアクセストークンを利用できます。ユーザープールの新機能を常に利用するには、ユーザープールのエッセンシャルプランを選択してください。

以下のセクションでは、エッセンシャルプランを使用してアプリケーションに追加できる機能を簡単に紹介します。詳細については、以降のページを参照してください。

その他のリソース

- アクセストークンのカスタマイズ: [トークン生成前の Lambda トリガー](#)
- E メール MFA: [SMS メッセージ MFA と E メールメッセージ MFA](#)
- パスワード履歴: [パスワード、アカウント復旧、パスワードポリシー](#)
- 拡張 UI: [マネージドログインページにブランディングを適用する](#)

トピック

- [アクセストークンのカスタマイズ](#)
- [E メール MFA](#)
- [パスワードの再利用防止](#)
- [マネージドログインのホストされたサインインおよび認可サーバー](#)
- [選択ベースの認証](#)

アクセストークンのカスタマイズ

ユーザープール[アクセストークン](#)は、[API へのアクセス](#)、[userInfo エンドポイント](#)からのユーザー属性の取得、または外部システムの[グループメンバーシップ](#)の確立に必要なアクセス許可をアプリケーションに付与します。高度なシナリオでは、実行時にアプリケーションが決定する追加の仮パラメータを、ユーザープールディレクトリのデフォルトのアクセストークンデータに追加することが考えられます。例えば、[Amazon Verified Permissions](#) を使用してユーザーの API アクセス許可を検証し、それに応じてアクセストークンのスコープを調整する場合です。

エッセンシャルプランは、[トークン生成前トリガー](#)の既存の機能に機能を追加します。下位の階層プランでは、追加のクレーム、ロール、グループメンバーシップを使用して ID トークンをカスタマイズできます。エッセンシャルでは、アクセストークンのクレーム、ロール、グループメンバーシップ、スコープをカスタマイズする、新しいバージョンのトリガー入カイベントを追加します。アクセストークンのカスタマイズは、イベントバージョン 3 で Machine to Machine (M2M) の[クライアント認証情報の付与](#)に利用できます。

アクセストークンをカスタマイズするには

1. エッセンシャル機能プランまたはプラス機能プランを選択します。
2. トリガーのための Lambda 関数を作成します。サンプル関数を使用するには、[Node.js に設定](#)します。
3. Lambda 関数に[サンプルコード](#)を入力するか、独自のコードを作成します。関数は、Amazon Cognito からのリクエストオブジェクトを処理し、含める変更を返す必要があります。
4. 新しい機能を[バージョン 2 または 3](#)のトークン生成前トリガーとして割り当てます。バージョン 2 のイベントは、ユーザー ID のアクセストークンをカスタマイズします。バージョン 3 は、ユーザー ID とマシン ID のアクセストークンをカスタマイズします。

詳細情報

- [アクセストークンのカスタマイズ](#)
- [Amazon Cognito ユーザープールでアクセストークンをカスタマイズする方法](#)

E メール MFA

Amazon Cognito ユーザープールは、多要素認証 (MFA) の 2 番目の要素として E メールを使用するように設定できます。E メール MFA を使用すると、Amazon Cognito は、認証プロセスを完了するために入力する必要がある検証コードを含む E メールをユーザーに送信できます。これにより、

ユーザーログインフローに重要なセキュリティレイヤーが追加されます。E メールベースの MFA を有効にするには、デフォルトの E メール設定ではなく、[Amazon SES E メール送信設定](#)を使用するようにユーザープールを設定する必要があります。

ユーザーが E メールメッセージによる MFA を選択すると、Amazon Cognito は、サインインの試行の都度、ユーザーの登録済み E メールアドレスに 1 回限りの検証コードを送信します。その後、認証フローを完了してアクセスするには、このコードをユーザープールに返す必要があります。これにより、ユーザーのユーザー名とパスワードが漏えいした場合でも、アプリケーションリソースにアクセスする前に、E メールで送信されたコードという追加の要素を提供する必要があるように設定されます。

詳細については、「[SMS メッセージ MFA と E メールメッセージ MFA](#)」を参照してください。以下は、E メール MFA でのユーザープールとユーザーを設定する方法の概要です。

Amazon Cognito コンソールで E メール MFA を設定するには

1. エッセンシャル機能プランまたはプラス機能プランを選択します。
2. ユーザープールの [サインイン] メニューで、[多要素認証] を編集します。
3. 設定する [MFA 強制適用] のレベルを選択します。[MFA を要求する] にすると、API のユーザーは MFA を使用して設定、確認、サインインのためのチャレンジを自動的に受け取ります。MFA を必要とするユーザープールでは、マネージドログインが MFA 要素を選択して設定するようユーザーに求めます。[オプションの MFA] にすると、アプリケーションは、MFA の設定と、E メール MFA のユーザー希望を設定するオプションをユーザーに提供する必要があります。
4. [MFA の方法] で、[E メールメッセージ] をオプションの 1 つとして選択します。

詳細情報

- [SMS メッセージ MFA と E メールメッセージ MFA](#)

パスワードの再利用防止

デフォルトでは、Amazon Cognito ユーザープールのパスワードポリシーは、パスワードの長さや文字タイプの要件、および仮パスワードの有効期限を設定します。エッセンシャルプランでは、パスワード履歴を強制する機能が追加されます。ユーザーがパスワードのリセットを試みると、ユーザープールは、ユーザーが以前のパスワードに設定できなくなるようにできます。パスワードポリシーの設定の詳細については、「[ユーザープールのパスワードの追加要件](#)」を参照してください。以下は、パスワード履歴ポリシーを使用してユーザープールを設定する方法の概要です。

Amazon Cognito コンソールでパスワード履歴を設定するには

1. エッセンシャル機能プランまたはプラス機能プランを選択します。
2. ユーザープールの [認証方法] メニューで、[パスワードポリシー] を見つけて [編集] を選択します。
3. 使用可能な他のオプションを設定し、[以前のパスワードの使用の防止] の値を設定します。

詳細情報

- [パスワード、アカウント復旧、パスワードポリシー](#)

マネージドログインのホストされたサインインおよび認可サーバー

Amazon Cognito ユーザープールには、OpenID Connect (OIDC) IdP、サードパーティ IdP のサービスプロバイダーまたは依拠しているパーティ、サインアップとサインインのための公開ユーザーインタラクティブページといった機能をサポートするオプションのウェブページがあります。これらのページは、まとめてマネージドログインと呼ばれます。ユーザープールのドメインを選択すると、Amazon Cognito は、これらのページを自動的にアクティブ化します。ライトプランでは、ホストされた UI が表示されます。エッセンシャルプランでは、この高度なバージョンのサインアップページとサインインページが表示されます。

マネージドログインページには、ブランディングとスタイルをカスタマイズするための、より多くの機能とオプションを備えたクリーンな最新のインターフェイスがあります。エッセンシャルプランは、マネージドログインへのアクセスを可能にする最下位のプランです。

Amazon Cognito コンソールでマネージドログインを設定するには

1. [設定] メニューで、エッセンシャル機能プランまたはプラス機能プランを選択します。
2. [ドメイン] メニューで、ユーザープールに [ドメインを割り当て](#)、[マネージドログイン] の [ブランディングバージョン] を選択します。
3. [マネージドログイン] メニューの [スタイル] タブで、[スタイルを作成] を選択して、スタイルをアプリケーションクライアントに割り当てるか、新しいアプリケーションクライアントを作成します。

詳細情報

- [ユーザープールのマネージドログイン](#)

選択ベースの認証

エッセンシャル階層は、拡張 UI および SDK ベースの API オペレーションに、認証オペレーション用の新しい認証フローを導入します。このフローは選択ベースの認証です。選択ベースの認証とは、ユーザーの認証の開始方法として、アプリケーション側でサインイン方法を宣言するのではなく、可能なサインイン方法をクエリして選択することから始める方法です。選択ベースの認証をサポートし、ユーザー名パスワード、パスワードなし、パスキーによる認証を有効にするようにユーザープールを設定できます。API の場合、これは USER_AUTH フローです。

Amazon Cognito コンソールで選択ベースの認証を設定するには

1. エッセンシャル機能プランまたはプラス機能プランを選択します。
2. ユーザープールの [サインイン] メニューで、[選択ベースのサインインのオプション] を編集します。選択ベースの認証で有効にする認証方法を選択して設定します。
3. ユーザープールの [認証方法] メニューで、サインインオペレーションの設定を編集します。

詳細情報

- [Amazon Cognito ユーザープールによる認証](#)

プラスプランの機能

プラス機能プランには、Amazon Cognito ユーザープールの高度なセキュリティ機能が含まれています。これらの機能は、実行時にユーザーコンテキストをログに記録して分析し、デバイス、場所、リクエストデータ、パスワードにセキュリティ上の問題が潜んでいないか確認します。次に、ユーザーアカウントをブロックまたはセキュリティ保護を追加する自動応答により、潜在的なリスクを軽減します。また、セキュリティログを Amazon S3、Amazon Data Firehose、または Amazon CloudWatch Logs にエクスポートして、さらに分析することもできます。

エッセンシャルプランからプラスプランに切り替えると、エッセンシャルのすべての機能に加えて、上位の機能を利用できます。例えば、脅威保護のセキュリティオプションセット (高度なセキュリティ機能とも呼ばれます) を利用できます。認証フロントエンドの脅威に自動的に適応するようにユーザープールを設定するには、ユーザープールのプラスプランを選択してください。

以下のセクションでは、プラスプランでアプリケーションに追加できる機能を簡単に紹介します。詳細については、以降のページを参照してください。

その他のリソース

- アダプティブ認証: [アダプティブ認証の使用](#)
- 漏えいした認証情報: [漏えいした認証情報の検出の使用](#)
- ログのエクスポート: [Amazon Cognito ユーザープールからのログのエクスポート](#)

トピック

- [脅威保護: アダプティブ認証](#)
- [脅威保護: 漏えいした認証情報の検出](#)
- [脅威保護: ユーザーアクティビティのログ記録](#)

脅威保護: アダプティブ認証

プラスプランには、アダプティブ認証機能が含まれています。この機能を有効にすると、ユーザープールはすべてのユーザー認証セッションのリスク評価を行います。結果として得られるリスク評価から、サインインしたユーザーのリスクレベルがしきい値の設定を超えている場合、認証をブロックしたり、MFA をプッシュしたりできます。アダプティブ認証を使用すると、ユーザープールとアプリケーションは、ユーザーのアカウントが攻撃を受けている可能性があるときに、自動的にブロックしたり、MFA を設定したりします。また、ユーザープールからリスク評価に関するフィードバックを提供して、将来の評価を調整することもできます。

Amazon Cognito コンソールでアダプティブ認証を設定するには

1. プラス機能プランを選択します。
2. ユーザープールの [脅威保護] メニューで、[脅威保護] の下にある [標準認証とカスタム認証] を編集します。
3. 標準認証またはカスタム認証の [強制モード] を [フル機能] に設定します。
4. [アダプティブ認証] で、さまざまなリスクレベルに対する自動リスク対応を設定します。

詳細情報

- [アダプティブ認証の使用](#)
- [アプリケーションにおける脅威保護のためのデータ収集](#)

脅威保護: 漏えいした認証情報の検出

プラスプランには、[漏えいした認証情報の検出] 機能が含まれています。この機能は、安全でないパスワードの使用や、それに伴って生じる意図しないアプリケーションアクセスの脅威から保護します。ユーザー名とパスワードによるサインインをユーザーに許可すると、他の場所で使用したパスワードが再利用される可能性があります。また、そのパスワードは漏洩しているか、一般的に推測されている可能性があります。漏えいした認証情報の検出では、ユーザーが送信したパスワードがユーザープールによって読み取られ、パスワードデータベースと比較されます。これにより、パスワードが漏えいしている可能性が高いと判断された場合は、サインインをブロックし、アプリケーションでユーザーのパスワードリセットを開始するようにユーザープールを設定できます。

漏えいした認証情報の検出は、新しいユーザーがサインアップしたとき、既存のユーザーがサインインしたとき、およびユーザーがパスワードをリセットしようとしたときに、安全でないパスワードに反応する可能性があります。この機能を使用すると、ユーザープールは、ユーザーがパスワードを入力する場所を問わず、安全でないパスワードを使用したサインインを防止または警告できます。

Amazon Cognito コンソールで漏えいした認証情報の検出を設定するには

1. プラス機能プランを選択します。
2. ユーザープールの [脅威保護] メニューで、[脅威保護] の下にある [標準認証とカスタム認証] を編集します。
3. 標準認証またはカスタム認証の [強制モード] を [フル機能] に設定します。
4. [漏えいした認証情報] で、チェックする認証オペレーションのタイプと、ユーザープールからの必要な自動レスポンスを設定します。

詳細情報

- [漏えいした認証情報の検出の使用](#)

脅威保護: ユーザーアクティビティのログ記録

プラスプランには、ユーザー認証試行のセキュリティ分析と詳細を示すログ記録機能が追加されています。アプリケーションに接続したデバイスに関するリスク評価、ユーザー IP アドレス、ユーザーエージェント、その他の情報を確認できます。これらの情報に基づいて、組み込みの脅威保護機能を使用して対処することも、独自のシステムでログを分析して適切な措置を講じることもできます。脅威保護から Amazon S3、CloudWatch Logs、または Amazon DynamoDB にログをエクスポートできます。

Amazon Cognito コンソールでユーザーアクティビティのログ記録を設定するには

1. プラス機能プランを選択します。
2. ユーザープールの [脅威保護] メニューで、[脅威保護] の下にある [標準認証とカスタム認証] を編集します。
3. 標準認証またはカスタム認証の [強制モード] を [監査のみ] に設定します。これはログに関する最小限の設定です。[フル機能] モードに切り替えて、その他の脅威保護機能を設定することもできます。
4. サードパーティー分析 AWS のサービス のためにログを別の にエクスポートするには、ユーザープールのログストリーミングメニューに移動し、エクスポート先を設定します。

詳細情報

- [ユーザー認証イベントのエクスポート](#)
- [Amazon Cognito ユーザープールからのログのエクスポート](#)

機能をオフにして機能プランを変更する

機能プランは、ユーザープールに設定オプションを追加します。これらの機能は、関連する機能プランがアクティブな場合にのみ設定して使用できます。例えば、アクセストークンのカスタマイズは、プラスプランとエッセンシャルプランでは設定できますが、ライトプランでは設定できません。これらの機能を非アクティブ化するには、アクティブな各コンポーネントを非アクティブ化する必要があります。Amazon Cognito コンソールの [設定] メニューの [切り替え] オプションは、機能プランを変更する前に非アクティブ化する必要がある機能を通知します。この章では、非アクティブ化によってユーザープール設定に加えられる変更と、これらの機能を個別にオフにする方法について説明します。

アクセストークンのカスタマイズ

アクセストークンのカスタマイズを含まないプランに切り替えるには、ユーザープールから [\[トークン生成前 Lambda トリガー\]](#) を削除する必要があります。アクセストークンをカスタマイズすることなく新しいトークン生成前トリガーを追加するには、トリガーに新しい関数を割り当て、V1_0 イベント用に設定します。これらのバージョン 1 トリガーイベントは、ID トークンの変更のみを処理できます。

アクセストークンのカスタマイズを手動で非アクティブ化するには、トークン生成前トリガーを削除し、新しいバージョン 1 のトリガーを追加します。

脅威保護

脅威保護のないプランに切り替えるには、ユーザープールの [脅威保護] メニューからすべての機能を非アクティブ化します。

ログのエクスポート

ログのエクスポートがないプランに切り替えるには、ユーザープールの [ログストリーミング] メニューで、この機能を非アクティブ化します。これにより、ユーザープールは、ローカルまたはエクスポートされたユーザーアクティビティログを生成しなくなります。UserActivity の EventSource 値がある設定を削除する [SetLogDeliveryConfiguration](#) API リクエストを送信することもできます。

E メール MFA

E メール MFA のないプランに切り替えるには、ユーザープールの [サインイン] メニューに移動します。[多要素認証] を編集し、使用可能な [MFA メソッド] の 1 つとしての [E メールメッセージ] を選択解除します。

Amazon Cognito ユーザープールに対するセキュリティのベストプラクティス

このページでは、一般的な脅威から保護する場合に実装できるセキュリティのベストプラクティスについて説明します。選択する設定は、各アプリケーションのユースケースによって異なります。管理オペレーションには、少なくとも最小特権を適用し、アプリケーションとユーザーのシークレットを保護する対策を講じることをお勧めします。もう 1 つの高度で効果的なステップは、AWS WAF ウェブ ACLs を設定してユーザープールに適用することです。

ネットワークレベルでユーザープールを保護する

AWS WAF ウェブ ACLs は、Amazon Cognito で構築する認証メカニズムのパフォーマンスとコストを保護できます。ウェブ ACL を使用すると、API およびマネージドログインのリクエストの前にガードレールを実装できます。ウェブ ACL は、ネットワーク層とアプリケーション層のフィルターを作成し、独自に作成したルールに基づいてトラフィックをドロップしたり、CAPTCHA を要求したりできます。リクエストは、ウェブ ACL ルールの条件を満たすまで Amazon Cognito リソースに渡されません。詳細については、「[AWS WAF ウェブ ACL](#)」を参照してください。

SMS メッセージの不正使用から保護する

ユーザープールでパブリックサインアップを許可すると、Amazon Cognito が SMS テキストメッセージで送信するコードを使用して、アカウント検証を設定できます。SMS メッセージは、不要なアクティビティに関連付けられ、AWS 請求額が増加する可能性があります。不正を伴うような SMS メッセージの送信に対して耐障害性のあるインフラストラクチャを設定してください。詳細については、AWS ブログの以下の投稿を参照してください。

- [Amazon Cognito ユーザープールを使用したユーザーサインアップ詐欺や SMS ポンピングのリスクを軽減する](#)
- [SMS Pumping に対する防御: AWS 人為的に拡張されたトラフィックとの戦いに役立つ新機能](#)

パブリック認証を理解する

Amazon Cognito ユーザープールには、カスタマーアイデンティティおよびアクセス管理 (CIAM) 機能があり、一般のユーザーがユーザーアカウントにサインアップしてアプリケーションにアクセスできるユースケースをサポートしています。ユーザープールでセルフサービスのサインアップを許可すると、パブリックインターネットからのユーザーアカウントへのリクエストが受け付けられます。セルフサービスリクエストは、[SignUp](#) や [InitiateAuth](#) などの API オペレーションからと、マネージドログインを使用したユーザーインタラクションから送信されます。パブリックリクエストに伴う不正使用を軽減するか、パブリック認証オペレーションを完全に無効化するようにユーザープールを設定できます。

以下の設定は、ユーザープールとアプリケーションクライアントでパブリック認証リクエストと内部認証リクエストを管理する方法の例をいくつか示しています。

ユーザープールへのパブリックアクセスに影響するユーザープール設定の例

設定	利用可能なオプション	設定場所	パブリック認証への影響	コンソール設定	API オペレーションとパラメータ
セルフサービスのサインアップ	アカウントへのサインアップや、管理者としてのユーザーアカウントの作成をユーザーに許可する	ユーザープール	パブリックサインアップを防止する	[サインアップ] – [セルフサービス	CreateUserPool 、 UpdateUserPool rPool

設定	利用可能なオプション	設定場所	パブリック認証への影響	コンソール設定	API オペレーションとパラメータ
				のサインアップ]	AdminCreateUserConfig – AllowAdminCreateUserOnly
管理者による確認	確認コードを新規ユーザーに送信するか、管理者に確認を求める	ユーザープール	管理者のアクションなしでのサインアップの確認を防止する	[サインアップ] – [Cognitoアシスト型の検証および確認]	CreateUserPool 、 UpdateUserPool AccountRecoverySettings – admin_only
ユーザー開示	サインイン時およびパスワードリセット時に「ユーザーが見つかりません」というメッセージを送信するか、開示を防止する	アプリケーションクライアント	サインイン名、Eメールアドレス、または電話番号の推測を防ぐ	[アプリケーションクライアント] – [ユーザー存在エラーを防ぐ]	CreateUserPoolClient 、 UpdateUserPoolClient PreventUserExistenceErrors

設定	利用可能なオプション	設定場所	パブリック認証への影響	コンソール設定	API オペレーションとパラメータ
クライアントシークレット	サインアップ時、サインイン時、パスワードリセット時にシークレットハッシュを要求するかどうかを指定する	アプリケーションクライアント	不正なソースからの認証リクエストを防ぐ	[アプリケーションクライアント] – [クライアントシークレット]	CreateUserPoolClient GenerateSecret
ウェブ ACL	認証リクエストのネットワークファイアウォールを有効化するかどうかを指定する	ユーザープール	管理者が定義したリクエスト特性と IP アドレスルールに基づいてアクセスを制限または禁止する	[AWS WAF] – [WAF 設定]	AssociateWebACL ResourceArn
外部 IdP	サードパーティ IdP、ユーザープールディレクトリ、または両方のユーザーによるサインインを許可する	アプリケーションクライアント	サインアップとサインインから ローカルユーザー または フェデレーションユーザー を除外する	[アプリケーションクライアント] – [ID プロバイダー]	CreateUserPoolClient 、 UpdateUserPoolClient SupportedIdentityProviders

設定	利用可能なオプション	設定場所	パブリック認証への影響	コンソール設定	API オペレーションとパラメータ
認可サーバー	認証用のパブリックウェブページをホストするかどうかを指定する	ユーザープール	パブリックウェブページをオフにし、SDK ベースの認証のみを許可する	[ドメイン]	CreateUserPoolDomain 任意のユーザープールドメインを作成すると、パブリックウェブページが使用可能になります。
脅威保護	悪意のあるアクティビティや安全でないパスワードの兆候に関するモニタリングを有効または無効にする	ユーザープールまたはアプリケーションクライアント	ユーザーが侵害の兆候を示した場合、自動的にサインインをブロックするか、MFA を要求することができる	[脅威保護] – [保護設定]	SetRiskConfiguration SetRiskConfiguration のパラメータは、脅威保護の設定を定義します。

秘密クライアントをクライアントシークレットで保護する

クライアントシークレットは、[アプリケーションクライアント](#)に関連付けられたオプションの文字列です。クライアントシークレットを持つアプリケーションクライアントへのすべての認証リクエストには、ユーザー名、クライアント ID、およびクライアントシークレットから生成された[シークレットハッシュ](#)を含める必要があります。クライアントシークレットを知らないユーザーは、最初からアプリケーションからシャットアウトされます。

ただし、クライアントシークレットには制限があります。クライアントシークレットをパブリッククライアントソフトウェアに埋め込むと、クライアントシークレットは閲覧可能になります。これにより、ユーザーの作成、パスワードリセットリクエストの送信、アプリケーションクライアントでの他のオペレーションの実行が可能になります。クライアントシークレットは、シークレットにアクセスできる唯一のエンティティがアプリケーションである場合にのみ実装する必要があります。通常、これが可能なのは、サーバー側の秘密クライアントアプリケーションです。これは、クライアントシークレットが必須である [M2M アプリケーション](#) にも当てはまります。暗号化されたローカルストレージまたは クラウドにクライアントシークレットを保存します AWS Secrets Manager。クライアントシークレットをパブリックインターネットに公開することは絶対に避けてください。

その他のシークレットを保護する

Amazon Cognito ユーザープールの認証システムでは、個人データ、パスワード、AWS 認証情報を扱う場合があります。アプリケーションでアクセスする可能性があるシークレットを扱うためのベストプラクティスを以下に示します。

パスワード

ユーザーは、アプリケーションへのサインイン時にパスワードを入力する場合があります。Amazon Cognito には、更新トークンがあり、これをアプリケーションで使用することで、期限切れのユーザーセッションを新しいパスワードの入力を求めることなく継続できます。パスワードやパスワードハッシュは、ローカルストレージに保存しないようにします。パスワードを不透明なものとして扱い、ユーザープールにのみ渡すようにアプリケーションを設計します。

ベストプラクティスとしては、[WebAuthn パスキー](#)を使用してパスワードなしの認証を実装します。パスワードを実装する必要がある場合は、[セキュアリモートパスワード \(SRP\) 認証フロー](#)と[多要素認証 \(MFA\)](#)を使用します。

AWS 認証情報

管理認証とユーザープール管理オペレーションには、AWS 認証情報を使用した認証が必要です。これらのオペレーションをアプリケーションに実装するには、[一時的な AWS 認証情報](#)への

安全なアクセスを許可します。認証情報へのアクセスは、管理するサーバーコンポーネントで実行するアプリケーションにのみ許可します。AWS 認証情報を持つアプリケーションを GitHub などのパブリックバージョン管理システムに配置しないでください。パブリッククライアント側のアプリケーションで AWS 認証情報をエンコードしないでください。

PKCE コード検証子

[Proof Key for Code Exchange \(PKCE\)](#) は、ユーザープール認可サーバーでの OpenID Connect (OIDC) 認証コード付与に使用します。アプリケーションは、認証コードをリクエストするときに、コード検証シークレットをユーザープールと共有します。認証コードをトークンと交換するには、クライアントがコード検証子を認識していることを再確認する必要があります。これにより、傍受された認証コードでトークンが発行されないようにします。

クライアントは、認可リクエストごとに新しいランダムなコード検証子を生成する必要があります。静的または予測可能なコード検証子を使用すると、攻撃者はハードコードされた検証子と認証コードを傍受するだけで済みます。コード検証子の値がユーザーに公開されないようにアプリケーションを設計します。

ユーザープール管理の最小特権

IAM ポリシーは、Amazon Cognito ユーザープールの管理および管理認証オペレーションに対してプリンシパルが持つアクセスレベルを定義できます。例えば、次のようになります。

- ウェブサーバーに対しては、管理 API オペレーションによる認証のためのアクセス許可を付与します。
- でユーザープールを管理する AWS IAM アイデンティティセンター ユーザーに AWS アカウント、ユーザープールのメンテナンスとレポートのためのアクセス許可を付与します。

Amazon Cognito でのリソースの詳細度は、IAM ポリシーの目的においては、ユーザープールとアイデンティティプールの [2つのリソースタイプ](#) に制限されています。個々のアプリケーションクライアントを管理するためのアクセス許可は適用できないことに注意してください。付与したアクセス許可はすべてのアプリケーションクライアントに有効であることを念頭にユーザープールを設定してください。組織に複数のアプリケーションテナントがあり、セキュリティモデルでテナント間の管理責任を分離する必要がある場合は、[ユーザープールごとのテナントを1つとするマルチテナンシー](#)を実装します。

ユーザー認証オペレーション (InitiateAuth など) のためのアクセス許可を持つ IAM ポリシーを作成することはできますが、これらのアクセス許可は効果がありません。[パブリック API オペレー](#)

[シジョンとトークンで認可された API オペレーション](#)は、IAM アクセス許可の対象ではありません。使用可能なユーザープール認証オペレーションのうち、アクセス許可を付与できるのは、管理サーバー側のオペレーション (AdminInitiateAuth など) に対してのみです。

最小特権 Action リストを使用して、ユーザープールの管理レベルを制限できます。次のポリシー例で対象としている管理者は、IdP、リソースサーバー、アプリケーションクライアント、ユーザープールドメインは管理できますが、ユーザーやユーザープールは管理できません。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UserPoolClientAdministrator",
      "Action": [
        "cognito-idp:CreateIdentityProvider",
        "cognito-idp:CreateManagedLoginBranding",
        "cognito-idp:CreateResourceServer",
        "cognito-idp:CreateUserPoolDomain",
        "cognito-idp>DeleteIdentityProvider",
        "cognito-idp>DeleteResourceServer",
        "cognito-idp>DeleteUserPoolDomain",
        "cognito-idp:DescribeIdentityProvider",
        "cognito-idp:DescribeManagedLoginBranding",
        "cognito-idp:DescribeManagedLoginBrandingByClient",
        "cognito-idp:DescribeResourceServer",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp:DescribeUserPoolDomain",
        "cognito-idp:GetIdentityProviderByIdentifier",
        "cognito-idp:GetUICustomization",
        "cognito-idp:ListIdentityProviders",
        "cognito-idp:ListResourceServers",
        "cognito-idp:ListUserPoolClients",
        "cognito-idp:ListUserPools",
        "cognito-idp:SetUICustomization",
        "cognito-idp:UpdateIdentityProvider",
        "cognito-idp:UpdateManagedLoginBranding",
        "cognito-idp:UpdateResourceServer",
        "cognito-idp:UpdateUserPoolClient",
        "cognito-idp:UpdateUserPoolDomain"
      ]
    }
  ]
}
```

```
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:cognito-idp:us-west-2:123456789012:userpool/us-
west-2_EXAMPLE"
  }
]
}
```

次のポリシー例では、サーバー側のアプリケーションに対して、ユーザーやグループの管理と認証を許可します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UserAdminAuthN",
      "Action": [
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminConfirmSignUp",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDeleteUserAttributes",
        "cognito-idp:AdminDisableProviderForUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminForgetDevice",
        "cognito-idp:AdminGetDevice",
        "cognito-idp:AdminGetUser",
        "cognito-idp:AdminInitiateAuth",
        "cognito-idp:AdminLinkProviderForUser",
        "cognito-idp:AdminListDevices",
        "cognito-idp:AdminListGroupsWithUser",
        "cognito-idp:AdminListUserAuthEvents",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp:AdminResetUserPassword",
        "cognito-idp:AdminRespondToAuthChallenge",
        "cognito-idp:AdminSetUserMFAPreference",
        "cognito-idp:AdminSetUserPassword",
        "cognito-idp:AdminSetUserSettings",

```

```
    "cognito-idp:AdminUpdateAuthEventFeedback",
    "cognito-idp:AdminUpdateDeviceStatus",
    "cognito-idp:AdminUpdateUserAttributes",
    "cognito-idp:AdminUserGlobalSignOut",
    "cognito-idp:AssociateSoftwareToken",
    "cognito-idp:ListGroups",
    "cognito-idp:ListUsers",
    "cognito-idp:ListUsersInGroup",
    "cognito-idp:RevokeToken",
    "cognito-idp:UpdateGroup",
    "cognito-idp:VerifySoftwareToken"
  ],
  "Effect": "Allow",
  "Resource": "arn:aws:cognito-idp:us-west-2:123456789012:userpool/us-
west-2_EXAMPLE"
}
```

トークンの保護と検証

トークンには、エンドユーザーに開示したくないグループメンバーシップやユーザー属性への内部参照が含まれている場合があります。ID トークンとアクセストークンは、ローカルストレージに保存しないでください。更新トークンは、ユーザープールのみがアクセスできるキーで暗号化され、ユーザーやアプリケーションからは不透明になります。ユーザーがサインアウトした場合や、セキュリティ上の理由でユーザーのセッションの保持が望ましくないと判断した場合は、[更新トークンの取り消し](#)を行います。

アクセストークンを使用すると、トークンが有効で期限切れでないことを独立して検証するシステムにのみ、アクセスを許可できます。検証リソースについては、「[JSON ウェブトークンの検証](#)」を参照してください。

信頼する ID プロバイダーを決定する

[SAML](#) または [OIDC](#) ID プロバイダー (IdP) を使用してユーザープールを設定すると、IdP は新しいユーザーを作成して、ユーザー属性を設定し、アプリケーションリソースにアクセスできます。SAML プロバイダーと OIDC プロバイダーは通常、企業間取引 (B2B) やエンタープライズのシナリオで使用され、お客様やお客様の直接の顧客がプロバイダーのメンバーシップと設定を管理します。

[ソーシャルプロバイダー](#)は、インターネット上の誰にでもユーザーアカウントを提供するため、エンタープライズプロバイダーと比べてお客様による管理が行き届きません。アプリケーションクライアントでソーシャル IdP を有効にするのは、一般ユーザーによるアプリケーションへのサインインとリソースへのアクセスを許可する準備が整った場合のみとします。

ユーザープロフィールへのアクセスに対するスコープの影響を理解する

ユーザープール認可サーバーへの認証リクエストで、アクセスコントロールのスコープをリクエストできます。これらのスコープは、外部リソースへのアクセスをユーザーに許可し、ユーザーが自身のユーザープロフィールを表示および変更するためのアクセスを許可できます。アプリケーションのオペレーションに必要な最小限のスコープをサポートするように、アプリケーションクライアントを設定します。

`aws.cognito.signin.user.admin` スコープは、[InitiateAuth](#) などのオペレーションで SDK 認証が発行するすべてのアクセストークンに存在します。スコープは、アプリケーションでのユーザープロフィールのセルフサービスオペレーション用に指定されています。このスコープは、認可サーバーに対してリクエストすることもできます。スコープは、[UpdateUserAttributes](#) や [GetUser](#) など、トークンで認可されたオペレーションに必須です。これらのオペレーションの影響は、アプリケーションクライアントの読み取りおよび書き込みアクセス許可によって制限されます。

`openid`、`profile`、`email`、`phone` の各スコープは、ユーザープール認可サーバーの [userInfo エンドポイント](#) へのリクエストを承認します。各スコープは、エンドポイントが返すことができる属性を定義します。`openid` スコープは、他のスコープを追加せずにリクエストすると、使用可能なすべての属性を返します。他のスコープを追加してリクエストすると、レスポンスは追加のスコープが示す属性に絞り込まれます。`openid` スコープは ID トークンのリクエストも示します。このスコープを [認可エンドポイント](#) へのリクエストから省略すると、Amazon Cognito はアクセストークンと、該当する場合は更新トークンのみを発行します。詳細については、「[アプリクライアントの用語](#)」の「OpenID Connect のスコープ」を参照してください。

ユーザー属性の入力をサニタイズする

配信方法やユーザー名となる可能性があるユーザー属性 (`email` など) には、[形式制限](#)があります。その他の属性には、文字列、ブール値、または数値データ型を含めることができます。文字列属性値は、さまざまな入力をサポートします。ユーザーディレクトリや、Amazon Cognito からユーザーに配信するメッセージに不要なデータを書き込む試みを防ぐようにアプリケーションを設定します。ユーザーが送信した文字列属性値は、アプリケーションにおいてクライアント側で検証してから Amazon Cognito に送信します。

ユーザープールは、指定した[属性マッピング](#)に基づいて、IdP からユーザープールに属性をマッピングします。安全で予測可能な IdP 属性のみをユーザープールの文字列属性にマッピングします。

Amazon Cognito ユーザープールによる認証

Amazon Cognito には、ユーザーを認証する方法がいくつか用意されています。ユーザーは、パスワードと WebAuthn パスキーを使用してサインインできます。Amazon Cognito は、E メールまたは SMS メッセージでワンタイムパスワードを送信できます。独自のチャレンジとレスポンスのシーケンスを調整する Lambda 関数を実装できます。これらは認証フローです。認証フローでは、ユーザーがシークレットを提供します。Amazon Cognito がシークレットを検証して JSON ウェブトークン (JWT) を発行し、これをアプリケーションが OIDC ライブラリで処理します。この章では、さまざまなアプリケーション環境でさまざまな認証フロー用にユーザープールとアプリケーションクライアントを設定する方法について説明します。マネージドログインのホストされたサインインページを使用するオプションと、AWS SDK で独自のロジックとフロントエンドを構築するためのオプションについて説明します。

ドメインの有無にかかわらず、すべてのユーザープールはユーザープール API でユーザーを認証できます。ユーザープールにドメインを追加すると、[ユーザープール エンドポイント](#)を使用できます。ユーザープール API は、API リクエストのさまざまな認可モデルとリクエストフローをサポートします。

ユーザーのアイデンティティを検証するために、Amazon Cognito では、E メールや SMS メッセージのワンタイムパスワードやパスキーなどのパスワードに加えて、チャレンジタイプを組み込んだ認証フローをサポートしています。

トピック

- [認証フローを実装する](#)
- [ユーザープールによる認証について知っておくべきこと](#)
- [認証セッションの例](#)
- [マネージドログインの認証方法を設定する](#)
- [AWS SDKsでの認証方法の管理](#)
- [認証フロー](#)
- [API 認証と SDK 認証の認可モデル](#)

認証フローを実装する

[マネージドログイン](#)を実装するか、認証用の AWS SDK [を使用してカスタム構築されたアプリケーションフロントエンド](#)を実装するかにかかわらず、実装する認証のタイプに合わせてアプリケーションクライアントを設定する必要があります。以下の情報では、[アプリケーションクライアント](#)とアプリケーションにおける認証フローの設定について説明します。

App client supported flows

Amazon Cognito コンソールまたは AWS SDK の API を使用して、アプリケーションクライアントでサポートされているフローを設定できます。これらのフローをサポートするようにアプリケーションクライアントを設定した後で、アプリケーションにデプロイできます。

次の手順では、Amazon Cognito コンソールを使用してアプリケーションクライアントで利用できる認証フローを設定します。

認証フロー用にアプリケーションクライアントを設定するには (コンソール)

1. にサインイン AWS し、[Amazon Cognito ユーザープールコンソール](#)に移動します。ユーザープールを選択するか、新しいユーザープールを作成します。
2. ユーザープールの設定で、[アプリケーションクライアント] メニューを選択します。アプリケーションクライアントを選択するか、新しく作成します。
3. [アプリケーションクライアントに関する情報] で、[編集] を選択します。
4. [アプリケーションクライアントのフロー] で、サポートする認証フローを選択します。

認証フロー用にアプリケーションクライアントを設定するには (API/SDK)

Amazon Cognito API を使用してアプリケーションクライアントで利用可能な認証フローを設定するには、[CreateUserPoolClient](#) リクエストまたは [UpdateUserPoolClient](#) リクエストで `ExplicitAuthFlows` の値を設定します。次の例では、セキュアリモートパスワード (SRP) と選択ベースの認証をクライアントにプロビジョニングします。

```
"ExplicitAuthFlows": [  
    "ALLOW_USER_AUTH",  
    "ALLOW_USER_SRP_AUTH"  
]
```

アプリケーションクライアントがサポートするフローを設定するときは、以下のオプションと API 値を指定できます。

アプリケーションクライアントでのフローのサポート

認証フロー	互換性	コンソール	API
選択ベースの認証	サーバー側、クライアント側	サインイン時に認証タイプを選択する	ALLOW_USE R_AUTH
永続的なパスワードによるサインイン	クライアント側	ユーザー名とパスワードでサインインする	ALLOW_USE R_PASSWORD_AUTH
永続的なパスワードと安全なペイロードによるサインイン	サーバー側、クライアント側	セキュアリモートパスワード (SRP) でサインインする	ALLOW_USE R_SRP_AUTH
トークンの更新	サーバー側、クライアント側	既存の認証済みセッションから新しいユーザートークンを取得する	ALLOW_REFRESH_TOKEN_AUTH
サーバー側の認証	サーバー側	サーバー側の管理認証情報でサインインする	ALLOW_ADMIN_USER_PASSWORD_AUTH
カスタム認証	サーバー側およびクライアント側のカスタム構築アプリケーション。マネージドログインとの互換性なし。	Lambda トリガーからのカスタム認証フローでサインインする	ALLOW_CUSTOM_AUTH

Implement flows in your application

マネージドログインでは、設定した認証オプションが、サインインページで自動的に利用可能になります。カスタム構築アプリケーションでは、初期フローの宣言から認証を開始します。

- ユーザーのフローオプションのリストから選択するには、USER_AUTH フローで[選択ベースの認証](#)を宣言します。このフローでは、[パスキー認証](#)や[パスワードなし](#)の認証など、クライアントベースの認証フローでは利用できない認証方法を利用できます。

- 認証フローを事前に選択するには、アプリケーションクライアントで利用可能な他のフローで [クライアントベースの認証](#) を宣言します。

ユーザーをサインインさせるときは、[InitiateAuth](#) リクエストまたは [AdminInitiateAuth](#) リクエストの本文に AuthFlow パラメータを含める必要があります。

選択ベースの認証:

```
"AuthFlow": "USER_AUTH"
```

SRP によるクライアントベースの認証:

```
"AuthFlow": "USER_SRP_AUTH"
```

ユーザープールによる認証について知っておくべきこと

Amazon Cognito ユーザープールを使用した認証モデルの設計では、以下の情報を考慮してください。

マネージドログインとホストされた UI の認証フロー

[マネージドログイン](#) には、クラシックのホストされた UI よりも多くの認証オプションがあります。例えば、ユーザーはマネージドログインでのみ、パスワードなしの認証とパスキー認証を実行できます。

AWS SDK 認証でのみ使用できるカスタム認証フロー

マネージドログインまたはクラシックのホストされた UI で、カスタム認証フローや [Lambda トリガーによるカスタム認証](#) を行うことはできません。カスタム認証は [AWS SDK による認証](#) で利用できます。

外部 ID プロバイダー (IdP) によるサインイン用のマネージドログイン

[AWS SDKs による認証](#) では、[サードパーティーの IdPs](#) を介してユーザーにサインインすることはできません。マネージドログインまたはクラシックのホストされた UI を実装し、IdP にリダイレクトしてから、結果の認証オブジェクトをアプリケーション内の OIDC ライブラリで処理する必要があります。マネージドログインの詳細については、「[ユーザープールのマネージドログイン](#)」を参照してください。

パスワードなしの認証が他のユーザー機能に与える影響

ユーザープールとアプリケーションクライアントで[ワンタイムパスワード](#)または[パスキー](#)を使用してパスワードなしのサインインをアクティブ化すると、ユーザーの作成と移行に影響します。パスワードなしのサインインがアクティブな場合:

1. 管理者はパスワードなしでユーザーを作成できます。デフォルトの招待メッセージテンプレートが変わり、{###} パスワードプレースホルダーを含まないようになります。詳細については、「[管理者としてのユーザーアカウントの作成](#)」を参照してください。
2. SDK ベースの[SignUp](#) オペレーションの場合、ユーザーはサインアップ時にパスワードを入力する必要はありません。マネージドログインとホストされた UI では、パスワードなしの認証が許可されている場合でも、サインアップページでパスワードが必要です。詳細については、「[ユーザーアカウントのサインアップと確認](#)」を参照してください。
3. CSV ファイルからインポートされたユーザーは、ユーザーの属性に、パスワードなしのサインインオプションに利用できる E メールアドレスまたは電話番号が含まれている場合、パスワードをリセットせずに、パスワードなしのオプションで即座にサインインできます。詳細については、「[CSV ファイルからユーザープールへのユーザーのインポート](#)」を参照してください。
4. パスワードなしの認証では、[ユーザー移行の Lambda トリガー](#)は呼び出されません。
5. パスワードなしの最初の要素でサインインしたユーザーは、セッションに[多要素認証 \(MFA\)](#) 要素を追加できません。MFA は、パスワードベースの認証フローでのみサポートされます。

パスキーに依拠しているパーティの URL をパブリックサフィックスリストに含めることはできない

所有しているドメイン名 (www.example.com など) を、パスキー設定で依拠しているパーティ (RP) の ID として使用できます。この設定は、所有しているドメインで実行するカスタム構築アプリケーションをサポートすることを目的としています。[パブリックサフィックスリスト](#) (PSL) には、保護された高レベルのドメインが含まれています。PSL のドメインに RP URL を設定しようとすると、Amazon Cognito はエラーを返します。

トピック

- [認証フローセッションの持続期間](#)
- [サインイン試行の失敗時におけるロックアウト動作](#)

認証フローセッションの持続期間

ユーザープールの機能によっては、アプリケーションが Amazon Cognito からトークンを取得する前に、InitiateAuth と RespondToAuthChallenge へのいくつかのチャレンジに応答することが必要になる場合があります。Amazon Cognito は、各リクエストへの応答にセッション文字列を含めます。API リクエストを 1 つの認証フローにまとめるには、前のリクエストに対する応答のセッション文字列を、後続の各リクエストに含めます。デフォルトでは、セッション文字列が期限切れになる前に、ユーザーに各チャレンジを完了するまでに 3 分間が与えられます。この期間を調整するには、アプリケーションクライアントの認証フローセッション持続期間を変更します。次の手順では、アプリクライアントの構成で、この設定を変更する方法について説明します。

Note

[認証フローセッションの持続期間] の設定は、Amazon Cognito ユーザープール API による認証に適用されます。マネージドログインの場合、多要素認証にはセッション時間が 3 分、パスワードリセットコードには 8 分が設定されます。

Amazon Cognito console

アプリクライアントの認証フローセッション持続期間を設定するには (AWS マネジメントコンソール)

1. ユーザープールの [App integration] (アプリの統合) タブで、[App clients and analytics] (アプリクライアントと分析) コンテナからアプリクライアントの名前を選択します。
2. [アプリケーションクライアントに関する情報] コンテナで [編集] を選択します。
3. [認証フローセッションの持続期間] の値を、SMS および E メール の MFA コードに必要な有効期間 (分単位) に変更します。これにより、ユーザーがアプリクライアントで認証チャレンジを完了するまでの時間も変更されます。
4. [Save changes] (変更の保存) をクリックします。

User pools API

アプリクライアントの認証フローセッション持続期間を設定するには (Amazon Cognito API)

1. DescribeUserPoolClient リクエストから既存のユーザープール設定を使用して UpdateUserPoolClient リクエストを準備します。UpdateUserPoolClient リクエストには、既存のアプリクライアントのプロパティをすべて含める必要があります。

2. AuthSessionValidity の値を、SMS MFA コードに必要な有効持続期間 (分単位) に変更します。これにより、ユーザーがアプリクライアントで認証チャレンジを完了するまでの時間も変更されます。

アプリクライアントの詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

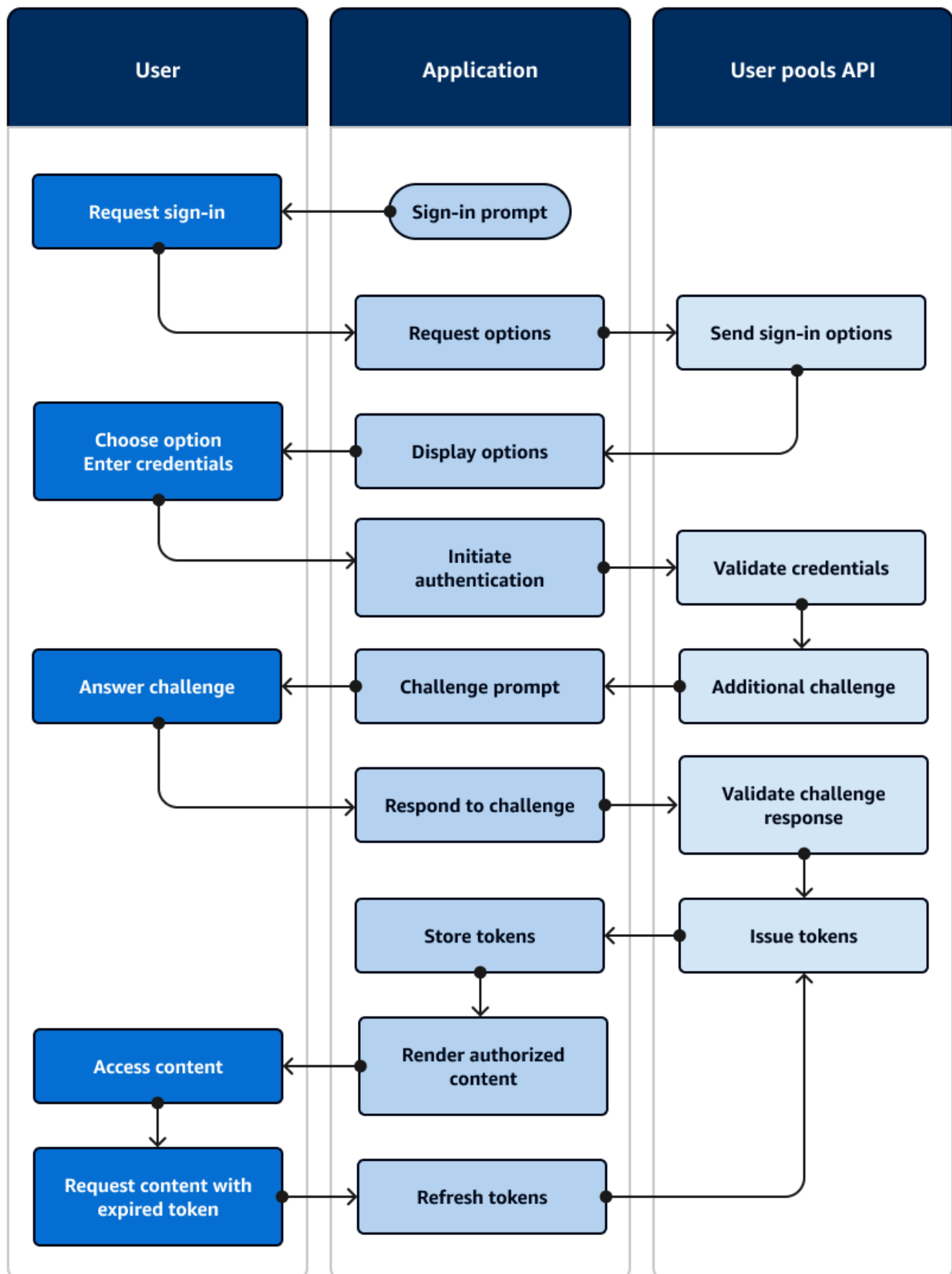
サインイン試行の失敗時におけるロックアウト動作

ユーザーのパスワードによるサインイン試行が 5 回失敗すると、認証されていない API オペレーションと IAM で認可された API オペレーションのどちらでリクエストしたかにかかわらず、Amazon Cognito はユーザーを 1 秒間ロックアウトします。ロックアウトの期間は、試行が 1 回失敗するたびに 2 倍になり、最大で約 15 分になります。

ロックアウト期間中に試行すると Password attempts exceeded 例外が生成され、その後のロックアウト期間の長さには影響しません。サインイン試行の累積失敗回数が n (Password attempts exceeded 例外を含まない) に達すると、Amazon Cognito はユーザーを $2^{(n-5)}$ 秒間ロックアウトします。ロックアウトを $n=0$ 初期状態にリセットするには、ユーザーは、ロックアウト期間後にサインインに成功するか、連続 15 分間、サインイン試行を開始してはなりません。この動作は変更される可能性があります。この動作は、パスワードベースの認証も実行しない限り、カスタムチャレンジに適用されません。

認証セッションの例

次の図とステップバイステップガイドは、ユーザーがアプリケーションにサインインする一般的なシナリオを示しています。サンプルアプリケーションでは、いくつかのサインインオプションをユーザーに示します。ユーザーは認証情報を入力してオプションを 1 つ選択し、追加の認証要素を指定してサインインします。



ユーザーがユーザー名とパスワードでサインインしたり、E メールメッセージでワンタイムコードをリクエストしたり、フィンガープリントオプションを選択したりできるサインインページを備えたアプリケーションを想像してください。

1. サインインプロンプト: アプリケーションは、[ログイン] ボタンがあるホーム画面を表示します。
2. サインインのリクエスト: ユーザーは [ログイン] を選択します。アプリケーションは、Cookie またはキャッシュからユーザー名を取得するか、ユーザー名の入力をユーザーに求めます。
3. リクエストオプション: アプリケーションは、USER_AUTH フローの InitiateAuth API リクエストでユーザーのサインインオプションをリクエストし、ユーザーが利用できるサインイン方法を求めます。
4. サインインオプションの送信: Amazon Cognito は、PASSWORD、EMAIL_OTP、WEB_AUTHN で応答します。応答には、次の応答で再生するためのセッション識別子が含まれます。
5. 表示オプション: アプリケーションは、ユーザーがユーザー名とパスワードの入力、ワンタイムコードの取得、フィンガープリントのスキャンを行うための UI 要素を表示します。
6. オプションの選択/認証情報の入力: ユーザーはユーザー名とパスワードを入力します。
7. 認証の開始: アプリケーションは、ユーザー名パスワードによるサインインを確認してユーザー名とパスワードを提供する RespondToAuthChallenge API リクエストにより、ユーザーのサインイン情報を提供します。
8. 認証情報の検証: Amazon Cognito はユーザーの認証情報を確認します。
9. 追加のチャレンジ: ユーザーは、Authenticator アプリケーションで多要素認証を設定しています。Amazon Cognito は SOFTWARE_TOKEN_MFA チャレンジを返します。
10. チャレンジプロンプト: アプリケーションは、ユーザーの Authenticator アプリケーションから時間ベースのワンタイムパスワード (TOTP) をリクエストするフォームを表示します。
11. チャレンジへの回答: ユーザーは TOTP を送信します。
12. チャレンジへの応答: 別の RespondToAuthChallenge リクエストで、アプリケーションはユーザーの TOTP を提供します。
13. チャレンジレスポンスの検証: Amazon Cognito はユーザーのコードを確認し、ユーザープールが現在のユーザーに追加のチャレンジを発行しないように設定されていることを検証します。
14. トークンの発行: Amazon Cognito は ID、アクセス、JSON ウェブトークン (JWT) の更新を返します。ユーザーの初期認証がここで完了します。
15. トークンの保存: アプリケーションはユーザーのトークンをキャッシュして、ユーザーデータの参照、リソースへのアクセスの許可、有効期限切れトークンの更新ができるようにします。

- 16 認可されたコンテンツのレンダリング: アプリケーションは、ユーザーのアイデンティティとロールに基づいて、リソースへのユーザーのアクセス権を決定し、アプリケーションコンテンツを配信します。
- 17 コンテンツへのアクセス: ユーザーはサインインし、アプリケーションの使用を開始します。
- 18 期限切れトークンによるコンテンツのリクエスト: 後で、ユーザーは認可を必要とするリソースをリクエストします。ユーザーのキャッシュされたトークンの有効期限が切れています。
- 19 更新トークン: アプリケーションは、ユーザーの保存された更新トークンを使用して `InitiateAuth` リクエストを行います。
- 20 トークンの発行: Amazon Cognito は新しい ID とアクセス用の JWT を返します。ユーザーのセッションは、追加の認証情報の入力を求められることなく、安全に更新されます。

[AWS Lambda トリガー](#)を使用して、ユーザーの認証方法をカスタマイズできます。トリガーは独自のチャレンジを認証フローの一部として発行し検証を行います。

また、安全なバックエンドサーバーでは、管理者の認証フローを使用することもできます。[ユーザー移行の認証フロー](#)を使用すると、ユーザーにパスワードのリセットを要求することなく、ユーザーを移行できます。

マネージドログインの認証方法を設定する

[マネージドログインページ](#)は、ユーザープール認証のウェブフロントエンドであり、ユーザーがサインイン、サインアウト、またはパスワードをリセットするとき呼び出すことができます。このモデルの場合、アプリケーションは OIDC ライブラリをインポートして、ユーザープールのマネージドログインページにより、ブラウザベースの認証試行を処理します。ユーザーが使用できる認証形式は、ユーザープールとアプリケーションクライアントの設定によって異なります。アプリケーションクライアントに `ALLOW_USER_AUTH` フローを実装すると、Amazon Cognito は使用可能なオプションからサインイン方法を選択するようユーザーに求めます。`ALLOW_USER_PASSWORD_AUTH` を実装して SAML プロバイダーを割り当てると、ログインページでユーザー名とパスワードを入力するか、IdP を介して接続するかを選択するオプションがユーザーに表示されます。

Amazon Cognito ユーザープールコンソールでは、アプリケーションのマネージドログイン認証の設定を開始できます。新しいユーザープールの作成時に開発対象のプラットフォームを指定すると、コンソールに OIDC および OAuth ライブラリの実装例と、サインインおよびサインアウトフローを実装するためのスターターコードが表示されます。マネージドログインは、多くの OIDC に依拠しているパーティの実装を使用して構築できます。可能な場合は、[OIDC に依拠しているパーティの認定ライブラリ](#)を使用することをお勧めします。詳細については、「[ユーザープールの開始方法](#)」を参照してください。

通常、OIDC に依拠しているパーティのライブラリは、ユーザープールの `.well-known/openid-configuration` エンドポイントを定期的にチェックして、トークンエンドポイントや承認エンドポイントなどの発行者 URL を特定します。ベストプラクティスとして、必要な場合は、この自動検出動作を実装します。発行者エンドポイントを手動で設定すると、エラーが発生する可能性があります。例えば、ユーザープールのドメインを変更するとします。openid-configuration へのパスは、ユーザープールのドメインにリンクされていないため、サービスエンドポイントを自動検出するアプリケーションは、ドメインの変更を自動的に検出します。

マネージドログインのユーザープール設定

アプリケーションの複数のプロバイダーによるサインインを許可したり、Amazon Cognito を独立したユーザーディレクトリとして使用したりする場合があります。また、ユーザー属性を収集したり、MFA をセットアップしてプロンプトを表示したり、E メールアドレスをユーザー名として要求したりする場合も考えられます。マネージドログインとホストされた UI のフィールドを直接編集することはできません。代わりに、ユーザープールの設定により、マネージドログインの認証フローの処理が自動的に設定されます。

ユーザープール設定の以下の項目により、Amazon Cognito がマネージドログインとホストされた UI でユーザーに提示する認証方法が決まります。

User pool options (Sign-in menu)

以下のオプションは、Amazon Cognito コンソールのユーザープールの [サインイン] メニューにあります。

Cognito ユーザープールのサインインオプション

ユーザー名のオプションがあります。マネージドログインページとホストされた UI ページは、選択した形式のユーザー名のみを受け入れます。例えば、E メールを唯一のサインインオプションとしてユーザープールを設定すると、マネージドログインページは E メール形式のユーザー名のみを受け入れます。

必須の属性

ユーザープールで属性を必須として設定すると、ユーザーはマネージドログインでのサインアップ時に、属性の値の入力を求められます。

選択ベースのサインインのオプション

[選択ベースの認証](#)での認証方法の設定があります。ここでは、[パスキー](#)や[パスワードなし](#)などの認証方法を有効または無効にできます。これらの方法は、[マネージドログインのドメイン](#)とライト階層より上の[機能プラン](#)を持つユーザープールでのみ使用できます。

多要素認証

マネージドログインとホストされた UI は、[MFA](#) の登録および認証オペレーションを処理します。ユーザープールで MFA が必須である場合、サインインページでは自動的に追加の要素を設定するようユーザーに求めます。また、MFA 設定を持つユーザーに MFA コードを使用して認証を完了するよう求めます。ユーザープールで MFA がオフまたはオプションである場合、サインインページでは MFA の設定を求めません。

ユーザーアカウントの復旧

ユーザープールのセルフサービスによる[アカウントの復旧](#)の設定により、ユーザーがパスワードをリセットできるリンクをサインインページに表示するかどうかが決まります。

User pool options (Domain menu)

以下のオプションは、Amazon Cognito コンソールのユーザープールの [ドメイン] メニューにあります。

ドメイン

ユーザープールのドメインを選択すると、認証のためにブラウザを起動したときにユーザーが開くリンクのパスが設定されます。

ブランディングバージョン

ブランディングバージョンを選択すると、ユーザープールのドメインにマネージドログインとホストされた UI のどちらが表示されるかが決まります。

User pool options (Social and external providers menu)

次のオプションは、Amazon Cognito コンソールのユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューにあります。

プロバイダー

ユーザープールに追加した ID プロバイダー (IdP) は、ユーザープール内の各アプリケーションクライアントに対してアクティブまたは非アクティブのままにすることができます。

App client options

以下のオプションは、Amazon Cognito コンソールのユーザープールの [アプリケーションクライアント] メニューにあります。これらのオプションを確認するには、リストからアプリケーションクライアントを選択します。

Quick Setup アップガイド

Quick Setup ガイドには、さまざまな開発者環境向けのコード例が記載されています。コード例には、マネージドログイン認証をアプリケーションと統合するために必要なライブラリが含まれています。

アプリケーションクライアント情報

この設定を編集して、現在のアプリケーションクライアントに対応するアプリケーションに割り当てられた IdP を設定します。Amazon Cognito は、マネージドログインページにユーザーのための選択肢を表示します。これらの選択肢は、割り当てられた方法と IdP によって決まります。例えば、MySAML という名前の SAML 2.0 IdP とローカルユーザープールのログインを割り当てると、マネージドログインページには認証方法プロンプトと MySAML のボタンが表示されます。

認証設定

この設定を編集して、アプリケーションの認証方法を設定します。Amazon Cognito は、マネージドログインページにユーザーのための選択肢を表示します。これらの選択肢は、IdP としてユーザープールを利用できるかどうかと、割り当てた方法によって決まります。例えば、選択ベースの ALLOW_USER_AUTH 認証を割り当てると、マネージドログインページには、E メールアドレスの入力やパスキーによるサインインなど、利用可能な選択肢が表示されます。マネージドログインページには、割り当てた IdP のボタンも表示されます。

ログインページ

このタブで利用可能なオプションを使用して、マネージドログインまたはホストされた UI のユーザーインタラクティブページの視覚的な効果を設定します。詳細については、「[マネージドログインページにブランディングを適用する](#)」を参照してください。

AWS SDKsでの認証方法の管理

Amazon Cognito ユーザープールのユーザーは、さまざまな初期サインインオプション (要素) を使用してサインインできます。一部の要素の場合、ユーザーは多要素認証 (MFA) を追加できます。これ

らの最初の要素として、ユーザー名とパスワード、ワンタイムパスワード、パスキー、カスタム認証などがあります。詳細については、「[認証フロー](#)」を参照してください。アプリケーションに組み込み UI コンポーネントがあり、AWS SDK モジュールをインポートする場合は、認証用のアプリケーションロジックを構築する必要があります。2 つの主要な方法のいずれかを選択し、その方法に基づいて、実装する認証メカニズムを選択する必要があります。

クライアントベースの認証を実装できます。この場合、アプリケーションまたはクライアントが認証タイプを事前に宣言します。別の選択肢は、選択ベースの認証です。この場合、アプリケーションがユーザー名を収集し、ユーザーが使用できる認証タイプをリクエストします。これらのモデルは、要件に応じて、同じアプリケーションにまとめて実装することも、複数のアプリケーションクライアントに分割して実装することもできます。方法ごとに機能が異なります。例えば、クライアントベースにはカスタム認証があり、選択ベースにはパスワードなしの認証があります。

ユーザープール API の AWS SDK 実装で認証を実行するカスタムビルドのアプリケーションでは、ユーザープール設定、アプリケーションクライアント設定、クライアント側の設定に合わせて API リクエストを構成する必要があります。USER_AUTH の AuthFlow で始まる InitiateAuth セッションは、選択ベースの認証を開始します。Amazon Cognito は、優先する認証方法または選択肢のリストのいずれかをチャレンジとして使用し、API に応答します。CUSTOM_AUTH の AuthFlow で始まるセッションは、Lambda トリガーによるカスタム認証に直接進みます。

一部の認証方法は 2 つのフロータイプのいずれかに固定されており、一部の方法は両方で使用できます。

トピック

- [選択ベースの認証](#)
- [クライアントベースの認証](#)

選択ベースの認証

アプリケーションは、選択ベースの認証として、以下の認証方法をリクエストできます。これらのオプションは、[InitiateAuth](#) または [AdminInitiateAuth](#) の PREFERRED_CHALLENGE パラメータで宣言するか、[RespondToAuthChallenge](#) または [AdminRespondToAuthChallenge](#) の ChallengeName パラメータで宣言します。

1. EMAIL_OTP および SMS_OTP

[ワンタイムパスワードによるパスワードなしのサインイン](#)

2. WEB_AUTHN

[WebAuthn パスキーを使用したパスワードなしのサインイン](#)

3. PASSWORD

[永続的なパスワードによるサインイン](#)

[永続的なパスワードと安全なペイロードによるサインイン](#)

[サインイン後の MFA](#)

これらのオプションを API コンテキストで確認するには、「[RespondToAuthChallenge](#)」で「ChallengeName」を参照してください。

選択ベースのサインインは、最初のリクエストに応じてチャレンジを発行します。このチャレンジは、リクエストされたオプションが利用可能であることを確認するか、利用可能な選択肢のリストを提供します。アプリケーションは、これらの選択肢をユーザーに表示することで、ユーザーが希望するサインイン方法の認証情報を入力し、チャレンジレスポンスで認証を続行できるようにします。

認証フローには、以下の選択ベースのオプションがあります。このタイプのすべてのリクエストでは、まずアプリケーションがユーザー名を収集するか、キャッシュから取得する必要があります。

1. USERNAME の AuthParameters のみを使用するオプションをリクエストします。Amazon Cognito は SELECT_CHALLENGE チャレンジを返します。そこから、アプリケーションは、ユーザーに対してチャレンジを選択し、このレスポンスをユーザープールに返すよう求めることができます。
2. PREFERRED_CHALLENGE の AuthParameters と優先チャレンジのパラメータ (ある場合) を使用して優先チャレンジをリクエストします。例えば、PASSWORD_SRP の PREFERRED_CHALLENGE をリクエストする場合は、SRP_A も含める必要があります。ユーザー、ユーザープール、アプリケーションクライアントがすべて優先チャレンジ用に設定されている場合、Amazon Cognito は、そのチャレンジの次のステップ (PASSWORD_SRP フローの PASSWORD_VERIFIER、EMAIL_OTP フリーや SMS_OTP フローの [CodeDeliveryDetails](#) など) を返します。優先チャレンジが利用できない場合、Amazon Cognito は SELECT_CHALLENGE と利用可能なチャレンジのリストを返します。
3. 最初にユーザーをサインインさせ、次に選択ベースの認証オプションをリクエストします。サインインしたユーザーのアクセストークンを含む [GetUserAuthFactors](#) リクエストは、利用可能な選択ベースの認証要素と MFA 設定を返します。このオプションを使用すると、ユーザーは最初にユーザー名とパスワードでサインインし、次に別の認証形式をアクティブ化できます。このオペ

レーションを使用して、優先チャレンジでサインインしたユーザーの追加のオプションを確認することもできます。

選択ベースの認証用に[アプリケーションクライアントを設定](#)するには、許可された認証フローに `ALLOW_USER_AUTH` を追加します。また、ユーザープール設定で許可する選択ベースの要素を選択する必要があります。以下のプロセスは、選択ベースの認証要素を選択する方法を示しています。

Amazon Cognito console

ユーザープールで選択ベースの認証オプションを設定するには

1. にサインイン AWS し、[Amazon Cognito ユーザープールコンソール](#)に移動します。ユーザープールを選択するか、新しいユーザープールを作成します。
2. ユーザープール設定で、[サインイン] メニューを選択します。[選択ベースのサインインのオプション] を見つけ、[編集] を選択します。
3. [パスワード] オプションは常に利用可能です。これには、PASSWORD フローと PASSWORD_SRP フローが含まれます。ユーザーのオプションに追加する [その他のオプション] を選択します。追加できるのは、パスキー (WEB_AUTHN)、メールメッセージのワンタイムパスワード (EMAIL_OTP)、SMS メッセージワンタイムパスワード (SMS_OTP) です。
4. [Save changes] (変更の保存) をクリックします。

API/SDK

次の [CreateUserPool](#) または [UpdateUserPool](#) リクエスト本文の一部は、選択ベースの認証に使用できるすべてのオプションを設定します。

```
"Policies": {
  "SignInPolicy": {
    "AllowedFirstAuthFactors": [
      "PASSWORD",
      "WEB_AUTHN",
      "EMAIL_OTP",
      "SMS_OTP"
    ]
  }
},
```

クライアントベースの認証

クライアントベースの認証は、以下の認証フローをサポートします。これらのオプションは、[InitiateAuth](#) または [AdminInitiateAuth](#) の AuthFlow パラメータで宣言します。

1. USER_PASSWORD_AUTH および ADMIN_USER_PASSWORD_AUTH

[永続的なパスワードによるサインイン](#)

[サインイン後の MFA](#)

この認証フローは、選択ベースの認証の PASSWORD と同等です。

2. USER_SRP_AUTH

[永続的なパスワードと安全なペイロードによるサインイン](#)

[サインイン後の MFA](#)

この認証フローは、選択ベースの認証の PASSWORD_SRP と同等です。

3. REFRESH_TOKEN_AUTH

[更新トークン](#)

この認証フローは、クライアントベースの認証でのみ使用できます。

4. CUSTOM_AUTH

[カスタム認証](#)

この認証フローは、クライアントベースの認証でのみ使用できます。

クライアントベースの認証の場合、Amazon Cognito は、ユーザーが認証フローを開始する前に認証方法を決定済みであると想定します。ユーザーが提供するサインイン要素を決定するロジックは、デフォルト設定またはカスタムプロンプトで決定し、ユーザープールへの最初のリクエストで宣言する必要があります。InitiateAuth リクエストは、リストされているオプションの 1 つ (USER_SRP_AUTH など) に直接対応するサインイン AuthFlow を宣言します。この宣言により、リクエストには、認証を開始するためのパラメータ (USERNAME、SECRET_HASH、SRP_A など) も含まれます。Amazon Cognito は、このリクエストに続けて、SRP の PASSWORD_VERIFIER や TOTP MFA によるパスワードサインインの SOFTWARE_TOKEN_MFA など、追加のチャレンジを発行する場合があります。

クライアントベースの認証用に[アプリケーションクライアントを設定](#)するには、許可された認証フローに ALLOW_USER_AUTH 以外の認証フローを追加します。例えば、ALLOW_USER_PASSWORD_AUTH、ALLOW_CUSTOM_AUTH、ALLOW_REFRESH_TOKEN_AUTH を追加します。クライアントベースの認証フローを許可するために、追加のユーザープール設定を行う必要はありません。

認証フロー

Amazon Cognito ユーザープールの認証プロセスは、ユーザーが最初の選択を行い、認証情報を送信し、追加のチャレンジに応答するフローとして最もよく説明できます。マネージドログイン認証をアプリケーションに実装すると、Amazon Cognito は、これらのプロンプトとチャレンジのフローを管理します。アプリケーションのバックエンドに AWS SDK を使用してフローを実装する場合は、リクエストのロジックを構築し、ユーザーに入力を求め、チャレンジに応答する必要があります。

アプリケーション管理者は、ユーザー特性、セキュリティ要件、認可モデルを利用して、ユーザーにサインインを許可する方法を決定します。以下の質問を自問してください。

- [他の ID プロバイダー \(IdP\)](#) の認証情報を使用してサインインすることをユーザーに許可するか？
- [ユーザー名とパスワード](#) は本人確認として十分であるか？
- ユーザー名パスワード認証のための認証リクエストが傍受される可能性はあるか？ アプリケーションでパスワードを送信したり、[ハッシュとソルトを使用して認証をネゴシエート](#)したりするか？
- パスワードの入力をスキップし、[ワンタイムパスワードを受信](#)してサインインすることをユーザーに許可するか？
- [サムプリント、顔、またはハードウェアセキュリティキー](#) を使用したサインインをユーザーに許可するか？
- どのような場合に [多要素認証 \(MFA\)](#) を必須にするか？
- [認証情報を再度要求せずにユーザーセッションを保持](#) するか？
- Amazon Cognito の組み込み機能を超えて [認可モデルを拡張](#) するか？

これらの質問に対する回答が得られたら、関連する機能をアクティブ化し、アプリケーションが行う認証リクエストで機能を実装する方法を検討できます。

ユーザーのサインインフローを設定したら、[GetUserAuthFactors](#) API オペレーションへのリクエストを使用して、MFA および[選択ベース](#)の認証要素の現在のステータスを確認できます。このオペレーションには、サインインしたユーザーのアクセストークンによる認可が必要です。このオペレーションからは、ユーザー認証要素と MFA 設定が返されます。

トピック

- [サードパーティの IdP によるサインイン](#)
- [永続的なパスワードによるサインイン](#)
- [永続的なパスワードと安全なペイロードによるサインイン](#)
- [ワンタイムパスワードによるパスワードなしのサインイン](#)
- [WebAuthn パスキーを使用したパスワードなしのサインイン](#)
- [サインイン後の MFA](#)
- [更新トークン](#)
- [カスタム認証](#)
- [ユーザー移行の認証フロー](#)

サードパーティの IdP によるサインイン

Amazon Cognito ユーザープールは、Apple でサインイン、Amazon でログイン、OpenID Connect (OIDC) サービスなど、IdP 間の認証セッションの中間ブローカーとして機能します。このプロセスは、フェデレーティッドサインインまたはフェデレーティッド認証とも呼ばれます。フェデレーティッド認証では、アプリケーションクライアントに組み込める認証フローは使用しません。代わりに、設定したユーザープール IdP をアプリケーションクライアントに割り当てます。フェデレーティッドサインインは、ユーザーがマネージドログインで IdP を選択するか、アプリケーションが IdP サインインページにリダイレクトするセッションを呼び出したときに発生します。

フェデレーティッドサインインでは、プライマリ認証要素と MFA 認証要素をユーザーの IdP に委任します。Amazon Cognito は、[ローカルユーザーにリンク](#)しない限り、このセクションの他の高度なフローをフェデレーションユーザーに追加しません。リンクされていないフェデレーションユーザーにはユーザー名がありますが、マッピングされた属性データのストアであり、通常はブラウザベースのフローから切り離してサインインに使用することはありません。

実装リソース

- [サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン](#)

永続的なパスワードによるサインイン

Amazon Cognito ユーザープールでは、すべてのユーザーにユーザー名があります。ユーザー名は、電話番号、E メールアドレス、選択した識別子、または管理者が指定した識別子です。このタイプ

のユーザーは、ユーザー名とパスワードでサインインし、必要に応じて MFA を指定できます。ユーザープールは、パブリックまたは IAM で認可された API オペレーションと SDK メソッドを使用してユーザー名パスワードによるサインインを実行できます。アプリケーションは、認証のためにパスワードをユーザープールに直接送信できます。ユーザープールは、認証が成功した結果として、追加のチャレンジまたは JSON ウェブトークン (JWT) で応答します。

Activate password sign-in

ユーザー名とパスワードによる [クライアントベースの認証](#) をアクティブ化するには、認証を許可するようにアプリケーションクライアントを設定します。Amazon Cognito コンソールで、ユーザープール設定の [アプリケーション] の下にある [アプリケーションクライアント] メニューに移動します。クライアント側のモバイルまたはネイティブアプリケーションでプレーンパスワードサインインを許可するには、アプリケーションクライアントを編集し、[認証フロー] で [ユーザー名とパスワード (ALLOW_USER_PASSWORD_AUTH)] を使用してサインインします] を選択します。サーバー側のアプリケーションでプレーンパスワードサインインを許可するには、アプリケーションクライアントを編集し、[サーバー側の管理者認証情報 (ALLOW_ADMIN_USER_PASSWORD_AUTH)] を使用してサインインします] を選択します。

ユーザー名とパスワードを使用して [選択ベースの認証](#) を有効にするには、それを許可するようにアプリケーションクライアントを設定します。アプリケーションクライアントを編集し、[選択ベースのサインイン: ALLOW_USER_AUTH] を選択します。

Edit app client information Info

App clients create integration between your app and your user pool. App clients can use their own subset of authentication flows, token characteristics, and security from your user pool.

App client

Configure app clients. App clients are the user pool authentication resources attached to your app. Select an app client to configure the permitted authentication actions for an app.

App client name Info

Enter a friendly name for your app client.

my-test-app-client

App client names are limited to 128 characters or less. Names may only contain alphanumeric characters, spaces, and the following special characters: + = , . @ -

Authentication flows Info

Choose authentication flows that your app will support. Refresh token authentication is always enabled. We have populated options based on your app type.

- ☒ **Choice-based sign-in: ALLOW_USER_AUTH**
Your user pool responds to sign-in requests with a list of available methods. Users can choose options like one-time passwords, biometric devices and security keys, and password-based sign-in with MFA.
- ☒ **Sign in with username and password: ALLOW_USER_PASSWORD_AUTH**
Users can sign in with a username and password. This method sends the username and password directly to your user pool.
- ☐ **Sign in with secure remote password (SRP): ALLOW_USER_SRP_AUTH**
Users can sign in with username and password. Your application uses SRP libraries in server-side or client-side sign-in operations to pass a password hash and verifier.
- ☒ **Sign in with server-side administrative credentials: ALLOW_ADMIN_USER_PASSWORD_AUTH**
Users can sign in with username and password in server-side authentication operations. This feature is not supported in HostedUI.
- ☐ **Sign in with custom authentication flows from Lambda triggers: ALLOW_CUSTOM_AUTH**
Users can sign in, optionally with username and password, and respond to custom challenges that you design in Lambda functions.
- ☒ **Get new user tokens from existing authenticated sessions: ALLOW_REFRESH_TOKEN_AUTH**
Your application can store a longer-lived refresh token that renews user sessions without additional user prompts.

選択ベースの認証フローでパスワード認証が使用可能であることを確認するには、[サインイン] メニューに移動し、[選択ベースのサインインのオプション] セクションを確認します。[使用でき

る選択肢] に [パスワード] が表示されている場合は、プレーンパスワード認証でサインインできます。[パスワード] オプションには、プレーンパスワード認証や SRP ユーザー名パスワード認証があります。

Edit options for choice-based sign-in [Info](#)

With the USER_AUTH sign-in flow, users can choose their primary sign-in factor from a list of options like password, passwordless, and passkey. Choose the types of authentication that you want to allow for users' first authentication prompt.

Available choices [Info](#)

Choose the types of authentication that you want to allow users to choose in the choice-based flow.

Enabled options

Password

Additional choices [Info](#)

Configure the authentication factors that you want users to be able to choose in prompt-based authentication. Users must register any factors that they want to choose for sign-in.

- ☐ Passkey
- ☐ Email message one-time password
- ☐ SMS message one-time password

[CreateUserPoolClient](#) リクエストまたは [UpdateUserPoolClient](#) リクエストで、希望するユーザー名パスワード認証オプションを使用して `ExplicitAuthFlows` を設定します。

```
"ExplicitAuthFlows": [
  "ALLOW_USER_PASSWORD_AUTH",
  "ALLOW_ADMIN_USER_PASSWORD_AUTH",
  "ALLOW_USER_AUTH"
]
```

[CreateUserPool](#) リクエストまたは [UpdateUserPool](#) リクエストで、サポートする選択ベースの認証フローを使用して `Policies` を設定します。AllowedFirstAuthFactors の PASSWORD 値には、認証フローとしてプレーンパスワードと SRP の両方のオプションが含まれます。

```
"Policies": {
  "SignInPolicy": {
    "AllowedFirstAuthFactors": [
      "PASSWORD",
      "EMAIL_OTP",
      "WEB_AUTHN"
    ]
  }
}
```

Choice-based sign-in with a password

ユーザー名パスワード認証を使用してユーザーをアプリケーションにサインインさせるには、[AdminInitiateAuth](#) リクエストまたは [InitiateAuth](#) リクエストの本文を、以下のように設定します。現在のユーザーがユーザー名パスワード認証の適格性を満たしている場合、このサインインリクエストは成功するか、次のチャレンジに進みます。それ以外の場合は、使用可能な [主な

要因] 認証チャレンジのリストで応答します。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

```
{
  "AuthFlow": "USER_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser",
    "PREFERRED_CHALLENGE" : "PASSWORD",
    "PASSWORD" : "[User's password]"
  },
  "ClientId": "1example23456789"
}
```

PREFERRED_CHALLENGE 値を省略して、ユーザーの適格なサインイン要素のリストを含むレスポンスを受け取ることもできます。

```
{
  "AuthFlow": "USER_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser"
  },
  "ClientId": "1example23456789"
}
```

優先チャレンジを送信しなかったか、送信先のユーザーが優先チャレンジの適格性を満たしていない場合、Amazon Cognito はオプションのリストを AvailableChallenges で返します。AvailableChallenges に PASSWORD の ChallengeName が含まれている場合、[RespondToAuthChallenge](#) または [AdminRespondToAuthChallenge](#) チャレンジレスポンスによる認証を次の形式で続行できます。チャレンジレスポンスを、最初のサインインリクエストへの API レスポンスに関連付ける Session パラメータを渡す必要があります。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

```
{
  "ChallengeName": "PASSWORD",
  "ChallengeResponses": {
    "USERNAME" : "testuser",
    "PASSWORD" : "[User's Password]"
  },
  "ClientId": "1example23456789",
  "Session": "[Session ID from the previous response]"
}
```

```
}
```

Amazon Cognito は、成功した適格な優先チャレンジリクエストと PASSWORD チャレンジレスポンスに対して、トークンを使用するか、多要素認証 (MFA) などの追加の必須チャレンジを使用して応答します。

Client-based sign-in with a password

ユーザー名パスワード認証を使用してクライアント側のアプリケーションにユーザーをサインインさせるには、[InitiateAuth](#) リクエストの本文を、以下のように設定します。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

```
{
  "AuthFlow": "USER_PASSWORD_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser",
    "PASSWORD" : "[User's password]"
  },
  "ClientId": "1example23456789"
}
```

ユーザー名パスワード認証を使用してサーバー側のアプリケーションにユーザーをサインインさせるには、[AdminInitiateAuth](#) リクエストの本文を、以下のように設定します。アプリケーションは、AWS 認証情報を使用してこのリクエストに署名する必要があります。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

```
{
  "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser",
    "PASSWORD" : "[User's password]"
  },
  "ClientId": "1example23456789"
}
```

Amazon Cognito は、トークンを使用するか、多要素認証 (MFA) などの追加の必須チャレンジを使用して、成功したリクエストに応答します。

永続的なパスワードと安全なペイロードによるサインイン

ユーザープールのユーザー名パスワードによるサインイン方法の別の形式として、セキユアリモートパスワード (SRP) プロトコルがあります。このオプションは、パスワードを知っていることの証明 (パスワードハッシュとソルト) を送信し、ユーザープールで検証できるようにします。Amazon Cognito へのリクエストには、読み取り可能なシークレット情報が含まれないため、ユーザーが入力したパスワードを処理するのはアプリケーションのみです。SRP 認証には、数学的計算が含まれていて、SDK にインポートできる既存のコンポーネントで処理するのが最適です。SRP は通常、モバイルアプリなどのクライアント側のアプリケーションに実装されます。プロトコルの詳細については、「[スタンフォード SRP ホームページ](#)」を参照してください。[ウィキペディア](#)にもリソースと例があります。認証フローの SRP 計算には、[さまざまなパブリックライブラリ](#)を利用できます。

Amazon Cognito 認証の開始/チャレンジ/応答シーケンスでは、SRP を使用してユーザーとパスワードを検証します。SRP 認証をサポートするようにユーザープールとアプリケーションクライアントを設定し、サインインリクエストとチャレンジレスポンスのロジックをアプリケーションに実装する必要があります。SRP ライブラリは、ユーザーのパスワードを所有していることをユーザープールに示すための乱数と計算値を生成できます。アプリケーションは、Amazon Cognito ユーザープールの API オペレーションと SDK メソッドにおいて、これらの計算値を JSON 形式の AuthParameters フィールドと ChallengeParameters フィールドに入力して検証を行います。

Activate SRP sign-in

ユーザー名と SRP を使用した[クライアントベースの認証](#)を有効にするには、認証を許可するようにアプリケーションクライアントを設定します。Amazon Cognito コンソールで、ユーザープール設定の [アプリケーション] の下にある [アプリケーションクライアント] メニューに移動します。クライアント側のモバイルアプリまたはネイティブアプリで SRP サインインを許可するには、アプリケーションクライアントを編集し、[認証フロー] で [セキユアリモートパスワード (SRP) (ALLOW_USER_SRP_AUTH)] を使用してサインインします] を選択します。

ユーザー名と SRP を使用した[選択ベースの認証](#)を有効にするには、アプリケーションクライアントを編集し、[選択ベースのサインイン: ALLOW_USER_AUTH] を選択します。

Edit app client information [Info](#)

App clients create integration between your app and your user pool. App clients can use their own subset of authentication flows, token characteristics, and security from your user pool.

App client

Configure app clients. App clients are the user pool authentication resources attached to your app. Select an app client to configure the permitted authentication actions for an app.

App client name [Info](#)

Enter a friendly name for your app client.

my-test-app-client

App client names are limited to 128 characters or less. Names may only contain alphanumeric characters, spaces, and the following special characters: + = , . @ -

Authentication flows [Info](#)

Choose authentication flows that your app will support. Refresh token authentication is always enabled. We have populated options based on your app type.

- ☒ **Choice-based sign-in: ALLOW_USER_AUTH**
Your user pool responds to sign-in requests with a list of available methods. Users can choose options like one-time passwords, biometric devices and security keys, and password-based sign-in with MFA.
- ☐ **Sign in with username and password: ALLOW_USER_PASSWORD_AUTH**
Users can sign in with a username and password. This method sends the username and password directly to your user pool.
- ☒ **Sign in with secure remote password (SRP): ALLOW_USER_SRP_AUTH**
Users can sign in with username and password. Your application uses SRP libraries in server-side or client-side sign-in operations to pass a password hash and verifier.
- ☐ **Sign in with server-side administrative credentials: ALLOW_ADMIN_USER_PASSWORD_AUTH**
Users can sign in with username and password in server-side authentication operations. This feature is not supported in HostedUI.
- ☐ **Sign in with custom authentication flows from Lambda triggers: ALLOW_CUSTOM_AUTH**
Users can sign in, optionally with username and password, and respond to custom challenges that you design in Lambda functions.
- ☒ **Get new user tokens from existing authenticated sessions: ALLOW_REFRESH_TOKEN_AUTH**
Your application can store a longer-lived refresh token that renews user sessions without additional user prompts.

選択ベースの認証フローで SRP 認証が使用可能であることを確認するには、[サインイン] メニューに移動し、[選択ベースのサインインのオプション] セクションを確認します。[使用できる選択肢] に [パスワード] が表示されている場合は、SRP 認証でサインインできます。[パスワード] オプションには、プレーンテキスト認証や SRP ユーザー名パスワード認証があります。

Edit options for choice-based sign-in [Info](#)

With the USER_AUTH sign-in flow, users can choose their primary sign-in factor from a list of options like password, passwordless, and passkey. Choose the types of authentication that you want to allow for users' first authentication prompt.

Available choices [Info](#)

Choose the types of authentication that you want to allow users to choose in the choice-based flow.

Enabled options

Password

Additional choices [Info](#)

Configure the authentication factors that you want users to be able to choose in prompt-based authentication. Users must register any factors that they want to choose for sign-in.

- ☐ Passkey
- ☐ Email message one-time password
- ☐ SMS message one-time password

[CreateUserPoolClient](#) リクエストまたは [UpdateUserPoolClient](#) リクエストで、希望するユーザー名パスワード認証オプションを使用して `ExplicitAuthFlows` を設定します。

```
"ExplicitAuthFlows": [
  "ALLOW_USER_SRP_AUTH",
  "ALLOW_USER_AUTH"
]
```

[CreateUserPool](#) リクエストまたは [UpdateUserPool](#) リクエストで、サポートする選択ベースの認証フローを使用して `Policies` を設定します。 `AllowedFirstAuthFactors` の `PASSWORD`

値には、認証フローとしてプレーンテキストパスワードと SRP の両方のオプションが含まれます。

```
"Policies": {
  "SignInPolicy": {
    "AllowedFirstAuthFactors": [
      "PASSWORD",
      "EMAIL_OTP",
      "WEB_AUTHN"
    ]
  }
}
```

Choice-based sign-in with SRP

SRP のユーザー名パスワード認証を使用してユーザーをアプリケーションにサインインさせるには、[AdminInitiateAuth](#) リクエストまたは [InitiateAuth](#) リクエストの本文を、以下のように設定します。現在のユーザーがユーザー名パスワード認証の適格性を満たしている場合、このサインインリクエストは成功するか、次のチャレンジに進みます。それ以外の場合は、使用可能な [主要因] 認証チャレンジのリストで応答します。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

```
{
  "AuthFlow": "USER_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser",
    "PREFERRED_CHALLENGE" : "PASSWORD_SRP",
    "SRP_A" : "[g^a % N]"
  },
  "ClientId": "1example23456789"
}
```

PREFERRED_CHALLENGE 値を省略して、ユーザーの適格なサインイン要素のリストを含むレスポンスを受け取ることもできます。

```
{
  "AuthFlow": "USER_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser"
  },
  "ClientId": "1example23456789"
}
```

```
}
```

優先チャレンジを送信しなかったか、送信先のユーザーが優先チャレンジの適格性を満たしていない場合、Amazon Cognito はオプションのリストを AvailableChallenges で返します。AvailableChallenges に PASSWORD_SRP の ChallengeName が含まれている場合、[RespondToAuthChallenge](#) または [AdminRespondToAuthChallenge](#) チャレンジレスポンスによる認証を次の形式で続行できます。チャレンジレスポンスを、最初のサインインリクエストへの API レスポンスに関連付ける Session パラメータを渡す必要があります。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

```
{
  "ChallengeName": "PASSWORD_SRP",
  "ChallengeResponses": {
    "USERNAME" : "testuser",
    "SRP_A" : "[g^a % N]"
  },
  "ClientId": "1example23456789",
  "Session": "[Session ID from the previous response]"
}
```

Amazon Cognito は、適格な優先チャレンジリクエストと PASSWORD_SRP チャレンジレスポンスに対して、PASSWORD_VERIFIER チャレンジで応答します。クライアントは SRP 計算を完了し、[RespondToAuthChallenge](#) リクエストまたは [AdminRespondToAuthChallenge](#) リクエストでチャレンジに応答する必要があります。

```
{
  "ChallengeName": "PASSWORD_VERIFIER",
  "ChallengeResponses": {
    "PASSWORD_CLAIM_SIGNATURE" : "string",
    "PASSWORD_CLAIM_SECRET_BLOCK" : "string",
    "TIMESTAMP" : "string"
  },
  "ClientId": "1example23456789",
  "Session": "[Session ID from the previous response]"
}
```

PASSWORD_VERIFIER チャレンジレスポンスが成功すると、Amazon Cognito はトークンを発行するか、多要素認証 (MFA) などの別の必要なチャレンジを発行します。

Client-based sign-in with SRP

SRP 認証は、サーバー側よりもクライアント側での認証の方が一般的です。ただし、[InitiateAuth](#) と [AdminInitiateAuth](#) では SRP 認証を使用できます。ユーザーをアプリケーションにサインインさせるには、InitiateAuth リクエストまたは AdminInitiateAuth リクエストの本文を、以下のように設定します。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

クライアントは、ジェネレータモジュール N_g をシークレットランダム整数の a 乗して SRP_A を生成します。

```
{
  "AuthFlow": "USER_SRP_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser",
    "SRP_A" : "[g^a % N]"
  },
  "ClientId": "1example23456789"
}
```

Amazon Cognito は PASSWORD_VERIFIER チャレンジで応答します。クライアントは SRP 計算を完了し、[RespondToAuthChallenge](#) リクエストまたは [AdminRespondToAuthChallenge](#) リクエストでチャレンジに応答する必要があります。

```
{
  "ChallengeName": "PASSWORD_VERIFIER",
  "ChallengeResponses": {
    "PASSWORD_CLAIM_SIGNATURE" : "string",
    "PASSWORD_CLAIM_SECRET_BLOCK" : "string",
    "TIMESTAMP" : "string"
  },
  "ClientId": "1example23456789",
  "Session": "[Session ID from the previous response]"
}
```

PASSWORD_VERIFIER チャレンジレスポンスが成功すると、Amazon Cognito はトークンを発行するか、多要素認証 (MFA) などの別の必要なチャレンジを発行します。

ワンタイムパスワードによるパスワードなしのサインイン

パスワードは紛失したり盗まれたりする場合があります。ユーザーが検証済みの E メールアドレス、電話番号、または Authenticator アプリケーションにのみアクセスできることを確認したい場合もあります。このような場合の解決策は、パスワードなしのサインインです。アプリケーションは、ユーザー名、E メールアドレス、または電話番号の入力をユーザーに求めることができます。これにより、Amazon Cognito はワンタイムパスワード (OTP) を生成し、ユーザーに確認を求めます。コードが正しければ、認証が完了します。

パスワードなしの認証フローは、ユーザープールで必須の多要素認証 (MFA) と互換性がありません。ユーザープールで MFA がオプションである場合、MFA をアクティブ化したユーザーはパスワードなしの最初の要素ではサインインできません。MFA オプションユーザープールに MFA 設定がないユーザーは、パスワードなしの各要素でサインインできます。詳細については、「[ユーザープールの MFA について知っておくべきこと](#)」を参照してください。

ユーザーがパスワードなしの認証の一環として、SMS や E メールメッセージで受信したコードを正しく入力すると、ユーザープールは、ユーザーを認証することに加えて、ユーザーの未検証の E メールアドレスや電話番号属性を検証済みとしてマークします。E メールアドレスや電話番号を[自動的に検証](#)するようにユーザープールを設定しているかどうかに関係なく、ユーザーステータスも UNCONFIRMED から CONFIRMED に変更されます。

パスワードなしのサインインの新しいオプション

ユーザープールでパスワードなしの認証を有効にすると、一部のユーザーフローの動作が変更されます。

1. ユーザーはパスワードなしでサインアップし、サインイン時にパスワードなしの要素を選択できます。管理者としてパスワードなしでユーザーを作成することもできます。
2. [CSV ファイルでインポート](#)したユーザーは、パスワードなしの要素を使用してすぐにサインインできます。サインイン前にパスワードを設定する必要はありません。
3. パスワードを持たないユーザーは、PreviousPassword パラメータなしで [ChangePassword](#) API リクエストを送信できます。

OTP による自動サインイン

E メール OTP または SMS メッセージ OTP でサインアップしてユーザーアカウントを確認するユーザーは、確認メッセージと一致するパスワードなしの要素を使用して自動的にサインインできます。マネージドログイン UI では、ユーザーがアカウントを確認して、確認コード配信方法による OTP サインインの適格性を満たしている場合、ユーザーは確認コードの入力後に、初回サインインに自動

的に進みます。AWS SDK を使用してカスタム構築されたアプリケーションで、以下のパラメータを [InitiateAuth](#) または [AdminInitiateAuth](#) オペレーションに渡します。

- Session リクエストパラメータとしての [ConfirmSignUp](#) API レスポンスの Session パラメータ。
- USER_AUTH の [AuthFlow](#)。

EMAIL_OTP または SMS_OTP の [PREFERRED_CHALLENGE](#) を渡すことができますが、必須ではありません。Session パラメータは認証の証明となり、有効なセッションコードを渡すと、Amazon Cognito は AuthParameters を無視します。

サインインオペレーションは、認証の成功を示すレスポンス [AuthenticationResult](#) を返します。以下の条件が満たされた場合、追加のチャレンジはありません。

- Session コードは有効であり、期限切れになっていない。
- ユーザーは OTP 認証方法の適格性を満たしている。

Activate passwordless sign-in

コンソール

パスワードなしのサインインを有効にするには、1 つ以上のパスワードなしのタイプでプライマリサインインを許可するようにユーザープールを設定し、USER_AUTH フローを許可するようにアプリケーションクライアントを設定します。Amazon Cognito コンソールで、ユーザープール設定の [認証] の下にある [サインイン] メニューに移動します。[選択ベースのサインインのオプション] を編集し、[メールメッセージのワンタイムパスワード] または [SMS メッセージワンタイムパスワード] を選択します。両方のオプションを有効にすることができます。変更内容を保存します。

[アプリケーションクライアント] メニューに移動し、アプリケーションクライアントを選択するか、新しいアプリケーションクライアントを作成します。[編集] を選択し、[サインイン時に認証タイプを選択: ALLOW_USER_AUTH] を選択します。

API/SDK

ユーザープール API で、[CreateUserPool](#) リクエストまたは [UpdateUserPool](#) リクエストで適切なパスワードなしのオプションを使用して SignInPolicy を設定します。

```
"SignInPolicy": {
```



```
"AllowedFirstAuthFactors": [
    "EMAIL_OTP",
    "SMS_OTP"
]
```

[CreateUserPoolClient](#) リクエストまたは [UpdateUserPoolClient](#) リクエストの必要なオプションを使用して、アプリケーションクライアント `ExplicitAuthFlows` を設定します。

```
"ExplicitAuthFlows": [
    "ALLOW_USER_AUTH"
]
```

Sign in with passwordless

パスワードなしのサインインには、[InitiateAuth](#) や [AdminInitiateAuth](#) で指定できる [クライアントベース](#) の AuthFlow はありません。OTP 認証は、USER_AUTH の [選択ベース](#) の AuthFlow でのみ使用可能であり、希望するサインインオプションをリクエストするか、ユーザーの [AvailableChallenges](#) からパスワードなしのオプションを選択することができます。ユーザーをアプリケーションにサインインさせるには、InitiateAuth リクエストまたは AdminInitiateAuth リクエストの本文を、以下のように設定します。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

この例では、ユーザーがどの方法でサインインするかわかりません。PREFERRED_CHALLENGE パラメータを追加し、優先チャレンジをユーザーに利用可能にすると、Amazon Cognito はそのチャレンジで応答します。

```
{
  "AuthFlow": "USER_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser"
  },
  "ClientId": "lexample23456789"
}
```

この例では、代わりに `"PREFERRED_CHALLENGE": "EMAIL_OTP"` または `"PREFERRED_CHALLENGE": "SMS_OTP"` を AuthParameters に追加できます。ユーザーがその優先方法の適格性を満たしている場合、ユーザープールはユーザーの E メールアドレスまたは電話番号に即座にコードを送信し、`"ChallengeName": "EMAIL_OTP"` または `"ChallengeName": "SMS_OTP"` を返します。

優先チャレンジを指定しない場合、Amazon Cognito は AvailableChallenges パラメータで応答します。

```
{
  "AvailableChallenges": [
    "EMAIL_OTP",
    "SMS_OTP",
    "PASSWORD"
  ],
  "Session": "[Session ID]"
}
```

このユーザーは、E メールメッセージの OTP、SMS メッセージの OTP、ユーザー名パスワードによるパスワードなしのサインインの適格性を満たしています。アプリケーションは、ユーザーに選択を求めるか、内部ロジックに基づいて選択を行うことができます。次に、[RespondToAuthChallenge](#) リクエストまたは [AdminRespondToAuthChallenge](#) リクエストに進み、チャレンジを選択できます。ユーザーが E メールメッセージの OTP を使用してパスワードなしの認証を完了したいとします。

```
{
  "ChallengeName": "SELECT_CHALLENGE",
  "ChallengeResponses": {
    "USERNAME" : "testuser",
    "ANSWER" : "EMAIL_OTP"
  },
  "ClientId": "lexample23456789",
  "Session": "[Session ID from the previous response]"
}
```

Amazon Cognito は EMAIL_OTP チャレンジで応答し、ユーザーの検証済み E メールアドレスにコードを送信します。アプリケーションは、このチャレンジに再度応答する必要があります。

EMAIL_OTP を PREFERRED_CHALLENGE としてリクエストした場合、これも次のチャレンジレスポンスとなります。

```
{
  "ChallengeName": "EMAIL_OTP",
  "ChallengeResponses": {
    "USERNAME" : "testuser",
    "EMAIL_OTP_CODE" : "123456"
  }
}
```

```
},  
"ClientId": "1example23456789",  
"Session": "[Session ID from the previous response]"  
}
```

WebAuthn パスキーを使用したパスワードなしのサインイン

パスキーは安全で、ユーザーの負担は比較的少なく済みます。パスキーサインインでは、ユーザーが認証に使用できる外部デバイスである Authenticator を利用します。通常のパスワードでは、ユーザーがフィッシング、パスワードの推測、認証情報の盗難などの脆弱性にさらされます。パスキーを使用すると、アプリケーションは、携帯電話やその他のデバイスで情報システムにアタッチまたは組み込まれている高度なセキュリティ対策を活用できます。一般的なパスキーサインインワークフローは、iOS キーチェーンや Google Chrome パスワードマネージャーなど、パスワードまたは認証情報マネージャーをデバイスで呼び出すことから始まります。デバイス上の認証情報マネージャーは、パスキーを選択して、既存の認証情報やデバイスロック解除メカニズムを使用してパスキーを許可するようユーザーに求めます。モダンな電話には、顔スキャナー、フィンガープリントスキャナー、ロック解除パターンなどのメカニズムが備わっており、いくつかは、強力な認証の要素である知識情報と所持情報を同時に満たします。バイオメトリクスを使用したパスキー認証の場合、パスキーは生体情報に該当します。

パスワードをサムプリント、顔、またはセキュリティキー認証に置き換えることもできます。これはパスキー認証または WebAuthn 認証と呼ばれます。アプリケーションのデベロッパーは、ユーザーが最初にパスワードでサインインした後に、生体認証デバイスの登録を許可するのが一般的です。Amazon Cognito ユーザープールを使用すると、アプリケーションは、このサインインオプションをユーザー向けに設定できます。パスキー認証は、多要素認証 (MFA) に対応していません。

パスワードなしの認証フローは、ユーザープールで必須の多要素認証 (MFA) と互換性がありません。ユーザープールで MFA がオプションである場合、MFA をアクティブ化したユーザーはパスワードなしの最初の要素ではサインインできません。MFA オプションユーザープールに MFA 設定がないユーザーは、パスワードなしの各要素でサインインできます。詳細については、「[ユーザープールの MFA について知っておくべきこと](#)」を参照してください。

パスキーとは

パスキーを使用すると、複雑なパスワードを覚えたり、OTP を入力したりする必要がなくなり、ユーザーエクスペリエンスが簡素化します。パスキーは、[ワールドワイドウェブコンソーシアム \(W3C\)](#) と FIDO (Fast Identity Online) アライアンスが策定した WebAuthn および CTAP2 標準に基づいています。ブラウザとプラットフォームは、これらの標準を実装し、ウェブアプリケーションまた

はモバイルアプリケーションでパスキーの登録や認証プロセスを開始するための API と、ユーザーがパスキー Authenticator を選択して操作するための UI を提供します。

ユーザーがウェブサイトやアプリに Authenticator を登録すると、Authenticator はパブリック/プライベートキーペアを作成します。WebAuthn ブラウザおよびプラットフォームは、パブリックキーをウェブサイトやアプリケーションのバックエンドに送信します。Authenticator は、ユーザーとアプリケーションに関するプライベートキー、キー ID、メタデータを保持します。ユーザーが登録済み Authenticator を使用して登録済みアプリケーションで認証しようとする、ランダムなチャレンジが生成されます。このチャレンジへの応答は、そのアプリケーションおよびユーザーの Authenticator のプライベートキーで生成されたチャレンジのデジタル署名および関連メタデータです。ブラウザまたはアプリケーションプラットフォームはデジタル署名を受け取り、アプリケーションのバックエンドに渡します。次に、アプリケーションは保存されたパブリックキーを使用して署名を検証します。

 Note

アプリケーションは、ユーザーから Authenticator に提供した認証シークレットや、プライベートキーに関する情報を受け取ることはありません。

現在市販されている Authenticator の例と機能を以下に示します。Authenticator は、これらのカテゴリのいずれかまたはすべてを満たしている場合があります。

- 一部の Authenticator は、PIN、顔やフィンガープリントによる生体認証入力、パスコードなどの要素を使用してユーザー検証を行ってからアクセスを許可し、正当なユーザーのみがアクションを承認できるようにします。一方、ユーザー検証機能を備えていない Authenticator や、ユーザー検証が要件でないときにユーザー検証をスキップできる Authenticator もあります。
- 一部の Authenticator (YubiKey ハードウェアトークンなど) はポータブルであり、USB、Bluetooth、または NFC 接続を介してデバイスと通信します。一部の Authenticator はローカルであり、PC の Windows Hello や iPhone の Face ID など、プラットフォームにバインドされています。デバイスにバインドされた Authenticator は、モバイルデバイスのように十分に小さい場合、ユーザーが持ち運ぶことができます。ワイヤレス通信を使用して、ハードウェア Authenticator をさまざまなプラットフォームに接続できる場合もあります。例えば、デスクトップブラウザのユーザーは、QR コードをスキャンするときに、スマートフォンをパスキー Authenticator として使用できます。
- 一部のプラットフォームにバインドされたパスキーはクラウドに同期されるため、複数の場所から使用できます。例えば、iPhone の Face ID パスキーは、パスキーメタデータを iCloud キー

チェーン内のユーザーの Apple アカウントと同期します。これらのパスキーは、ユーザーが各デバイスを個別に登録する必要がなく、Apple デバイス間でのシームレスな認証を許可します。1Password、Dashlane、Bitwarden などのソフトウェアベースの Authenticator は、ユーザーがこれをインストールしたすべてのプラットフォームでパスキーを同期します。

WebAuthn の用語では、ウェブサイトとアプリは依拠しているパーティです。各パスキーは、特定の依拠しているパーティの ID に関連付けられます。この ID は、パスキー認証を受け入れるウェブサイトやアプリを表す統合識別子となります。開発者は、適切な認証範囲を確保するために、依拠しているパーティの ID を慎重に選択する必要があります。一般的な依拠しているパーティの ID は、ウェブサーバーのルートドメイン名です。この依拠しているパーティの ID 仕様を持つパスキーは、該当するドメインとサブドメインに対して認証できます。ユーザーがアクセスするウェブサイトの URL が依拠しているパーティの ID と一致しない場合、ブラウザとプラットフォームはパスキー認証を拒否します。同様に、モバイルアプリの場合、パスキーを使用できるのは、.well-known 関連付けファイルにアプリケーションパスが存在する場合のみです。このファイルは、依拠しているパーティの ID が示すパスにおいてアプリケーションが使用可能にします。

パスキーは検出可能です。ユーザーがユーザー名を入力することなく、ブラウザまたはプラットフォームで自動的に認識して使用できます。ユーザーは、パスキー認証をサポートするウェブサイトやアプリにアクセスすると、ブラウザまたはプラットフォームが既に認識しているパスキーのリストから選択できます。QR コードをスキャンすることもできます。

Amazon Cognito でパスキー認証を実装する方法

パスキーは、ライトを除くすべての[機能プラン](#)で利用できるオプトイン機能です。[選択ベースの認証フロー](#)でのみ使用できます。[マネージドログイン](#)では、Amazon Cognito がパスキー認証のロジックを処理します。[AWS SDK で Amazon Cognito ユーザープールの API](#)を使用して、アプリケーションのバックエンドでパスキー認証を行うこともできます。

Amazon Cognito は、ES256 (-7) と RS256 (-257) の 2 つの非対称暗号化アルゴリズムのいずれかを使用して作成されたパスキーを認識します。ほとんどの Authenticator は両方のアルゴリズムをサポートしています。デフォルトでは、ユーザーはハードウェアトークン、モバイルスマートフォン、ソフトウェア Authenticator アプリケーションなど、あらゆるタイプの認証を設定できます。Amazon Cognito は現在、[証明](#)の強制をサポートしていません。

ユーザープールでは、ユーザー検証を [優先] または [必須] に設定できます。この設定は、値を指定しない API リクエストではデフォルトで [優先] になり、Amazon Cognito コンソールではデフォルトで [優先] が選択されます。ユーザー検証を [優先] に設定すると、ユーザーはユーザー検証機能を

持たない Authenticator を設定でき、登録および認証オペレーションはユーザー検証なしで成功します。パスキーの登録と認証でユーザー検証を義務付けるには、この設定を [必須] に変更します。

パスキー設定で設定した依拠しているパーティ (RP) の ID は重要な決定事項です。特に指定せず、[ドメインブランディングバージョン](#)がマネージドログインの場合、ユーザープールはデフォルトで[カスタムドメイン](#)の名前を RP ID として想定します。カスタムドメインがなく、特に指定しない場合、ユーザープールはデフォルトで[プレフィックスドメイン](#)の RP ID になります。パブリックサフィックスリスト (PSL) にないドメイン名に RP ID を設定することもできます。RP ID エントリは、マネージドログインと SDK 認証のパスキー登録と認証に適用されます。パスキーは、Amazon Cognito が RP ID をドメインとする .well-known 関連付けファイルを見つけることができるモバイルアプリケーションでのみ機能します。ベストプラクティスとして、ウェブサイトまたはアプリが公開される前に、依拠しているパーティの ID の値を決定して設定します。RP ID を変更すると、ユーザーは新しい RP ID を使用して再登録する必要があります。

各ユーザーは最大 20 個のパスキーを登録できます。パスキーを登録できるのは、少なくとも 1 回ユーザープールにサインインした後のみです。マネージドログインは、パスキー登録の労力を大幅に軽減します。ユーザープールとアプリケーションクライアントのパスキー認証を有効にすると、マネージドログインドメインを持つユーザープールは、エンドユーザーが新しいユーザーアカウントにサインアップした後にパスキーを登録するよう通知します。また、ユーザーのブラウザをいつでも呼び出して、パスキー登録用のマネージドログインページに誘導することもできます。Amazon Cognito がパスキー認証を開始する前に、ユーザーはユーザー名を入力する必要があります。マネージドログインは、これを自動的に処理します。サインインページでユーザー名を入力を求め、ユーザーが少なくとも 1 つのパスキーを登録していることを検証し、パスキーを使用してサインインするよう促します。同様に、SDK ベースのアプリケーションは、ユーザー名を入力を求め、認証リクエストでユーザー名を提供する必要があります。

パスキーを使用したユーザープール認証を設定したときに、カスタムドメインとプレフィックスドメインがある場合、RP ID はデフォルトでカスタムドメインの完全修飾ドメイン名 (FQDN) になります。Amazon Cognito コンソールでプレフィックスドメインを RP ID として設定するには、カスタムドメインを削除するか、プレフィックスドメインの FQDN をサードパーティドメインとして入力します。

Activate passkey sign-in

コンソール

パスキーによるサインインを有効にするには、1 つ以上のパスワードなしのタイプでプライマリサインインを許可するようにユーザープールを設定し、さらに USER_AUTH フローを許可するようにアプリケーションクライアントを設定します。Amazon Cognito コンソールで、ユーザー

プール設定の [認証] の下にある [サインイン] メニューに移動します。[選択ベースのサインインのオプション] を編集し、[利用できる選択肢] のリストに [パスキー] を追加します。

[認証方法] メニューに移動し、[パスキー] を編集します。

- [ユーザー検証] は、現在のユーザーがパスキーの使用を許可されていることを追加でチェックするために、ユーザープールでパスキーデバイスを必須にするかどうかの設定です。ユーザー検証を使用してデバイスを設定するようユーザーに推奨するが、必須にしない場合は、[優先] を選択します。ユーザー検証を必要とするデバイスのみをサポートする場合は、[必須] を選択します。詳細については、w3.org で「[ユーザー検証](#)」を参照してください。
- [依拠しているパーティーの ID のドメイン] は、アプリケーションからユーザーのパスキー登録リクエストに渡す識別子です。ユーザーのパスキー発行者と信頼関係を持つ対象を設定します。依拠しているパーティーの ID は、以下の場合に、ユーザープールのドメインになります。

Cognito ドメイン

ユーザープールの Amazon Cognito [プレフィックスドメイン](#)。

カスタムドメイン

ユーザープールの[カスタムドメイン](#)。

サードパーティードメイン

ユーザープールのマネージドログインページを使用しないアプリケーションのドメイン。この設定は通常、[ドメイン](#)を持たないユーザープールに関連付けられ、バックエンドで AWS SDK とユーザープール API を使用して認証を実行します。

[アプリケーションクライアント] メニューに移動し、アプリケーションクライアントを選択するか、新しいアプリケーションクライアントを作成します。[編集] を選択し、[認証フロー] で [サインイン時に認証タイプを選択: ALLOW_USER_AUTH] を選択します。

API/SDK

ユーザープール API で、[CreateUserPool](#) リクエストまたは [UpdateUserPool](#) リクエストの適切なパスキーオプションを使用して SignInPolicy を設定します。パスキー認証の WEB_AUTHN オプションには、他のオプションの少なくとも 1 つを追加する必要があります。パスキー登録には、既存の認証セッションが必要です。

```
"SignInPolicy": {  
  "AllowedFirstAuthFactors": [  

```



```
    "PASSWORD",  
    "WEB_AUTHN"  
  ]  
}
```

ユーザー検証の設定と RP ID を [SetUserPoolMfaConfig](#) リクエストの `WebAuthnConfiguration` パラメータに指定します。パスキー認証結果の対象となる `RelyingPartyId` は、ユーザープールのプレフィックスドメインまたはカスタムドメインにするか、独自に選択したドメインにすることができます。

```
"WebAuthnConfiguration": {  
  "RelyingPartyId": "example.auth.us-east-1.amazoncognito.com",  
  "UserVerification": "preferred"  
}
```

[CreateUserPoolClient](#) リクエストまたは [UpdateUserPoolClient](#) リクエストの必要なオプションを使用して、アプリケーションクライアント `ExplicitAuthFlows` を設定します。

```
"ExplicitAuthFlows": [  
  "ALLOW_USER_AUTH"  
]
```

Register a passkey (managed login)

マネージドログインは、ユーザーのパスキー登録を処理します。ユーザープールでパスキー認証が有効な場合、Amazon Cognito は新規ユーザーアカウントの登録時にパスキーを設定するようユーザーに求めます。

ユーザーがサインアップ済みでパスキーを設定していないか、アカウントが管理者権限で作成されている場合、Amazon Cognito はパスキーを設定するようユーザーに促しません。この状態のユーザーは、パスキーを登録する前に、パスワードやパスワードなしの OTP など、別の要素でサインインする必要があります。

パスキーを登録するには

1. [サインインページ](#)にユーザーを誘導します。

```
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. ユーザーからの認証結果を処理します。この例の場合、Amazon Cognito は、アプリケーションがトークンと交換する認証コードを使用して、ユーザーを `www.example.com` にリダイレクトします。
3. ユーザーをパスキー登録ページに誘導します。ユーザーには、ブラウザ Cookie があり、サインインセッションを維持します。パスキー URL は `client_id` パラメータと `redirect_uri` パラメータを使用します。Amazon Cognito は、認証されたユーザーにのみ、このページへのアクセスを許可します。パスワード、E メール OTP、または SMS OTP を使用してユーザーをサインインさせ、次のパターンに一致する URL を呼び出します。

このリクエストに、`response_type` や `scope` など、[認可エンドポイント](#)の他のパラメータを追加することもできます。

```
https://auth.example.com/passkeys/add?  
client_id=1example23456789&redirect_uri=https%3A%2F%2Fwww.example.com
```

Register a passkey (SDK)

メタデータを使用してパスキーの認証情報を [PublicKeyCreationOptions](#) オブジェクトに登録します。このオブジェクトは、サインインしたユーザーの認証情報を使用して生成し、API リクエストでパスキー発行者に提示できます。発行者は、[RegistrationResponseJSON](#) オブジェクトを返して、パスキーの登録を確認します。

パスキー登録プロセスを開始するには、既存のサインインオプションを使用してユーザーをサインインさせます。現在のユーザーのアクセストークンを使用して、[トークンで認可されたStartWebAuthnRegistration](#) API リクエストを承認します。次に示すのは、`GetWebAuthnRegistrationOptions` リクエストの本文の例です。

```
{  
  "AccessToken": "eyJra456defEXAMPLE"  
}
```

ユーザープールからのレスポンスには、`PublicKeyCreationOptions` オブジェクトが含まれています。このオブジェクトを API リクエストでユーザーの発行者に提示します。オブジェクトは、パブリックキーや依拠しているパーティの ID などの情報を提供します。発行者は `RegistrationResponseJSON` オブジェクトで応答します。

[CompleteWebAuthnRegistration](#) API リクエストで (ユーザーのアクセストークンで再認可された) 登録レスポンスを提示します。ユーザープールから空の本文を持つ HTTP 200 レスポンスが返されると、ユーザーのパスキーが登録されています。

Sign in with a passkey

パスワードなしのサインインには、[InitiateAuth](#) および [AdminInitiateAuth](#) で指定できる AuthFlow はありません。代わりに、USER_AUTH の AuthFlow を宣言してサインインオプションをリクエストするか、ユーザープールからのレスポンスでパスワードなしのオプションを選択する必要があります。ユーザーをアプリケーションにサインインさせるには、InitiateAuth リクエストまたは AdminInitiateAuth リクエストの本文を、以下のように設定します。このパラメータセットは、サインインに必要な最小限のものです。その他のパラメータも使用可能です。

この例では、ユーザーがパスキーでサインインすることがわかっているため、PREFERRED_CHALLENGE パラメータを追加します。

```
{
  "AuthFlow": "USER_AUTH",
  "AuthParameters": {
    "USERNAME" : "testuser",
    "PREFERRED_CHALLENGE" : "WEB_AUTHN"
  },
  "ClientId": "lexample23456789"
}
```

Amazon Cognito は WEB_AUTHN チャレンジで応答します。アプリケーションは、このチャレンジに応答する必要があります。ユーザーのパスキープロバイダーを使用してサインインリクエストを開始します。[AuthenticationResponseJSON](#) オブジェクトが返されます。

```
{
  "ChallengeName": "WEB_AUTHN",
  "ChallengeResponses": {
    "USERNAME" : "testuser",
    "CREDENTIAL" : "{AuthenticationResponseJSON}"
  },
  "ClientId": "lexample23456789",
  "Session": "[Session ID from the previous response]"
}
```

サインイン後の MFA

ユーザー名パスワードフローでサインインを完了したユーザーに対して、E メールメッセージ、SMS メッセージ、またはコード生成アプリケーションのワンタイムパスワードを使用した追加の検証を求めるように設定できます。MFA は、ワンタイムパスワードを使用する最初の認証要素であるパスワードなしのサインインや、MFA を含まない WebAuthn パスキーとは異なります。ユーザープールの MFA はチャレンジレスポンスモデルであり、ユーザーは最初にパスワードを知っていることを証明し、次に登録された第 2 要素デバイスにアクセスできることを証明します。

実装リソース

- [ユーザープールに MFA を追加します](#)

更新トークン

認証情報を再入力せずにユーザーがサインインしたままにする場合、更新トークンは、アプリケーションがユーザーのセッションを維持するためのツールとなります。アプリケーションは、更新トークンをユーザープールに提示し、新しい ID トークンやアクセストークンと交換できます。トークンの更新により、サインインしているユーザーが引き続きアクティブであることの確認、更新された属性情報の取得、ユーザーの介入なしのアクセスコントロール権限の更新を行うことができます。

実装リソース

- [更新トークン](#)

カスタム認証

ここに記載していない、ユーザーの認証方法を設定することもできます。そのためには、Lambda トリガーによるカスタム認証を使用します。一連の Lambda 関数により、Amazon Cognito はチャレンジを発行し、ユーザーに回答を求める質問を行い、回答が正確であることを確認し、別のチャレンジを発行すべきかどうかを判断します。質問と回答には、セキュリティの質問、CAPTCHA サービスへのリクエスト、外部 MFA サービス API へのリクエスト、またはこれらすべてが順に含まれます。

実装リソース

- [カスタム認証チャレンジの Lambda トリガー](#)

カスタム認証フロー

Amazon Cognito ユーザープールにより、カスタム認証フローを使用することもできます。これは AWS Lambda トリガーを使用したチャレンジ/レスポンスベースの認証モデルの作成に役立ちます。

カスタム認証フローを使用すると、チャレンジとレスポンスサイクルをカスタマイズすることが可能になり、さまざまな要件に対応できます。このフローは、使用する認証のタイプを示す `InitiateAuth` API 操作の呼び出しから始まり、初期の認証パラメータを提供します。Amazon Cognito は、以下のいずれかの情報を使用して `InitiateAuth` コールに応答します。

- ユーザーへのチャレンジ、ならびにセッションおよびパラメータ。
- ユーザーが認証に失敗した場合はエラー。
- `InitiateAuth` コールで渡されたパラメータがユーザーのサインインに十分な場合は、ID、アクセス権限、更新トークン。(通常、ユーザーまたはアプリケーションは最初にチャレンジに応答する必要がありますが、カスタムコードはこれを判定する必要があります。)

Amazon Cognito がチャレンジで `InitiateAuth` コールに応答する場合、アプリは追加の入力を収集して、`RespondToAuthChallenge` 操作を呼び出します。このコールは、チャレンジ応答を提供し、セッションを返します。Amazon Cognito は、`RespondToAuthChallenge` コールに対しても `InitiateAuth` コール同様の対応をします。ユーザーがサインインしている場合、Amazon Cognito はトークンを提供します。ユーザーがサインインしていない場合、Amazon Cognito は別のチャレンジまたはエラーを提供します。Amazon Cognito が別のチャレンジを返した場合、ユーザーが正常にサインインするかエラーが返されるまで、シーケンスが繰り返され、アプリは `RespondToAuthChallenge` を呼び出します。`InitiateAuth` API 操作と `RespondToAuthChallenge` API 操作に関する詳細については、[API ドキュメント](#)に記載されています。

カスタム認証フローとチャレンジ

アプリはカスタム認証フローを開始するために、`InitiateAuth` を `CUSTOM_AUTH` として `Authflow` を呼び出すことができます。カスタム認証フローで、3 つの Lambda トリガーがチャレンジとレスポンスの検証を制御します。

- `DefineAuthChallenge` Lambda トリガーは、以前のチャレンジとレスポンスのセッション配列を入力として使用します。そして、次のチャレンジ名と、ユーザーが認証済みで、トークンを付与できるかどうかを示すブール値を生成します。この Lambda トリガーは、チャレンジを通じてユーザーのパスを制御する状態マシンです。

- `CreateAuthChallenge` Lambda トリガーは、チャレンジ名を入力として使用し、チャレンジとパラメータを生成してレスポンスを評価します。`DefineAuthChallenge` が次のチャレンジとして `CUSTOM_CHALLENGE` を返すと、認証フローは、`CreateAuthChallenge` を呼び出します。`CreateAuthChallenge` Lambda トリガーは、チャレンジメタデータパラメータで次のタイプのチャレンジを渡します。
- `VerifyAuthChallengeResponse` Lambda 関数は、レスポンスを評価し、レスポンスが有効であるかどうかを示すブール値を返します。

カスタム認証フローでは、SRP パスワードの検証や SMS を介した MFA などの、内部定義されたチャレンジを組み合わせることもできます。CAPTCHA や秘密の質問などのカスタムチャレンジを使用できます。

カスタム認証フローにおける SRP パスワードの検証の使用

カスタム認証フローに SRP を含める場合には、最初に SRP を処理する必要があります。

- カスタムフローで SRP パスワードの検証を開始する場合、アプリは `InitiateAuth` を `CUSTOM_AUTH` として `Authflow` を呼び出します。`AuthParameters` マップで、アプリケーションからのリクエストは、`SRP_A`: (SRP A 値) および `CHALLENGE_NAME`: `SRP_A` を含んでいます。
- `CUSTOM_AUTH` フローは、`challengeName`: `SRP_A` および `challengeResult`: `true` の初期セッションにより、`DefineAuthChallenge` の Lambda トリガーを呼び出します。Lambda 関数は、`challengeName`: `PASSWORD_VERIFIER`、`issueTokens`: `false`、および `failAuthentication`: `false` により、応答します。
- 次にアプリケーションは、(`challengeName`: `PASSWORD_VERIFIER`、そして `challengeResponses` マップ内の SRP に必要な他のパラメータを使用して) `RespondToAuthChallenge` を呼び出す必要があります。
- Amazon Cognito がパスワードを検証すると、`challengeName`: `PASSWORD_VERIFIER` と `challengeResult`: `true` の 2 番目のセッションで、`RespondToAuthChallenge` により Lambda トリガーの `DefineAuthChallenge` が呼び出されます。その時点で `DefineAuthChallenge` Lambda トリガーは、`challengeName`: `CUSTOM_CHALLENGE` を使用して応答することでカスタムチャレンジを開始します。
- MFA がユーザーに対して有効になっている場合、Amazon Cognito がパスワードを確認した後、ユーザーは MFA の設定またはサインインを要求されます。

Note

Amazon Cognito がホストするサインインウェブページは、[カスタム認証チャレンジの Lambda トリガー](#) をアクティブ化できません。

サンプルコードを含めた Lambda トリガーの詳細については、「[Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#)」を参照してください。

ユーザー移行の認証フロー

ユーザー移行用 Lambda トリガーは、レガシーのユーザー管理システムからユーザープールにユーザーを移行する際に役立ちます。USER_PASSWORD_AUTH 認証フローを選択している場合には、移行中のユーザーがパスワードをリセットする必要はありません。このフローは、認証中に暗号化された SSL 接続を介してユーザーのパスワードをサービスに送信します。

すべてのユーザーの移行が完了したら、フローをよりセキュアな SRP フローに切り替えます。SRP フローは、ネットワーク経由でパスワードを送信しません。

Lambda トリガーの詳細については、「[Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#)」を参照してください。

Lambda トリガーを使用したユーザーの移行の詳細については、「[ユーザー移行の Lambda トリガーを使用したユーザーのインポート](#)」を参照してください。

API 認証と SDK 認証の認可モデル

ユーザープール認証を使用してアプリケーション開発を始める場合は、アプリケーションタイプに適した API 認可モデルを決める必要があります。認可モデルとは、Amazon Cognito ユーザープール API および SDK 統合の認証コンポーネントを使用してリクエストを行うための認証を提供するシステムです。Amazon Cognito には、IAM 認可、パブリック、トークン認可の 3 つの承認モデルがあります。

IAM で認可されたリクエストの場合、認可は、リクエストの Authorization ヘッダーにある一連の AWS IAM 認証情報による署名から取得されます。サーバー側のアプリケーションの場合、この方法は IAM 認可を使用して認証オペレーションを保護します。パブリック (認証されていない) 認証リクエストでは、認可は必要ありません。これは、ユーザーに配布されるクライアント側のアプリケーションに適しています。トークンで認可されたオペレーションは、通常、パブリックオペレーションと組み合わせて実装され、認可は、リクエストの Authorization ヘッダーに含まれるセッション

トークンまたはアクセストークンから取得されます。Amazon Cognito 認証では、通常、2 つ以上の API オペレーションを順番に実装する必要があります。使用する API オペレーションはアプリケーションの特性によって異なります。アプリケーションがユーザーに配布されるパブリッククライアントでは、サインインのリクエストに認可を必要としないパブリックオペレーションを使用します。トークンで認可されたオペレーションは、パブリックアプリケーションでユーザーのセッションを続行します。アプリケーションロジックがリモートシステムでホストされているサーバー側のクライアントでは、サインインリクエストの IAM 認可を使用して認証オペレーションを保護します。以下の API オペレーションペアとそれに対応する SDK メソッドは、使用可能な認可モデルに対応しています。

各パブリック認証オペレーションには、[UpdateUserAttributes](#) や [AdminUpdateUserAttributes](#) など、サーバー側に同等の形式があります。クライアント側のオペレーションはユーザーが開始し、確認を必要としますが、サーバー側のオペレーションでは、変更がユーザープール管理者によってコミットされたと思われ、変更がすぐに有効になります。この例では、Amazon Cognito からユーザーに確認コードを記載したメッセージを送信し、ユーザーのアクセストークンで [VerifyUserAttribute](#) リクエストによるコードの送信を許可します。サーバー側のアプリケーションは、任意の属性の値を即座に設定できますが、サインインに E メールアドレスと電話番号を使用する場合、これらの値の変更に [特別な考慮事項が適用](#)されます。

API 認証を比較し、API オペレーションと認可モデルの詳細なリストを確認するには、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

Client-side (public) authentication

以下は、クライアント側のアプリケーションでの一般的なリクエストのシーケンスです。

1. パブリック [InitiateAuth](#) オペレーションは、ユーザー名とパスワードなどのプライマリ認証情報を送信します。
2. トークンで認可された [RespondToAuthChallenge](#) オペレーションは、InitiateAuth レスポンスからのセッショントークンと、MFA などのチャレンジに対する回答を送信します。セッショントークン認可は、まだ完了していない認証サイクルの一部であるリクエストを示します。
3. トークンで認可された [ConfirmDevice](#) オペレーションは、アクセストークンを送信し、記憶されたデバイスをユーザーのプロファイルに追加する書き込みオペレーションを実行します。アクセストークン認可は、ユーザーが認証を完了した後のセルフサービスオペレーションに対するリクエストを示します。

詳細については、「[クライアント側の認証オプション](#)」および「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

Server-side authentication

以下は、サーバー側のオペレーションからの一般的なリクエストのシーケンスです。各リクエストには、アプリケーションサーバーに発行された IAM マシン認証情報で署名された [AWS Signature Version 4](#) の認可ヘッダーがあります。

1. [AdminInitiateAuth](#) オペレーションは、ユーザー名とパスワードなどのプライマリ認証情報を送信します。
2. [AdminRespondToAuthChallenge](#) オペレーションは、MFA などのチャレンジに対する回答を送信します。
3. [AdminUpdateDeviceStatus](#) オペレーションは、AdminInitiateAuth [レスポンス](#) のデバイスキーを記憶済みとして設定します。

詳細については、「[サーバー側の認証オプション](#)」および「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

認証が失敗するか、Amazon Cognito がユーザーにトークンを発行するまで、ユーザーは連続してチャレンジに回答して認証を進めます。カスタム認証フローをサポートするために、さまざまなチャレンジを含むプロセスで Amazon Cognito でこれらの手順を繰り返すことができます。

トピック

- [サーバー側の認証オプション](#)
- [クライアント側の認証オプション](#)
- [API、OIDC、マネージドログインページの認証についての理解](#)
- [認可モデル別にグループ化された API オペレーションのリスト](#)

サーバー側の認証オプション

ウェブアプリケーションなどのサーバー側のアプリケーションは、ブラウザや SSH セッションなどのリモートディスプレイアプリケーションでクライアントがロードする認証をリモートサーバーに実装します。サーバー側のアプリケーションには通常、以下の特性があります。

- Java、Ruby、Node.js などの言語でサーバーにインストールされたアプリケーションに組み込まれています。
- ユーザープールの [アプリケーションクライアント](#) に接続します。アプリケーションクライアントは、秘密クライアントと呼ばれるクライアントシークレットを持つ場合があります。

- AWS 認証情報にアクセスできます。
- 認証のために[マネージドログイン](#)を呼び出すか、AWS SDK を使用してユーザープール API で IAM で認可されたオペレーションを使用します。
- 内部のユーザーにサービスを提供するだけでなく、一般のユーザーにサービスを提供する場合もあります。

ユーザープール API を使用したサーバー側のオペレーションでは、パスワード、ワンタイムパスワード、またはパスキーを主要なサインイン要素として使用できます。サーバー側のアプリでのユーザープール認証は、クライアント側のアプリでの認証に似ています。次の点が異なります。

- サーバー側のアプリケーションは [AdminInitiateAuth](#) API リクエストを行います。このオペレーションには、cognito-idp:AdminInitiateAuthおよび を含むアクセス許可を持つ AWS 認証情報が必要ですcognito-idp:AdminRespondToAuthChallenge。オペレーションは、必要なチャレンジまたは認証結果を返します。
- アプリケーションはチャレンジを受信すると、[AdminRespondToAuthChallenge](#) API リクエストを行います。AdminRespondToAuthChallenge API オペレーションには AWS 認証情報も必要です。

AWS 認証情報を使用した Amazon Cognito API リクエストの署名の詳細については、AWS 全般のリファレンス[の署名バージョン 4 の署名プロセス](#)を参照してください。

AdminInitiateAuth からのレスポンス ChallengeParameters では、USER_ID_FOR_SRP 属性 (存在する場合) に、ユーザーのエイリアス (E メールアドレスや電話番号など) ではなく、実際のユーザー名が含まれています。AdminRespondToAuthChallenge への呼び出しの ChallengeResponses では、このユーザー名を USERNAME パラメータで渡す必要があります。

Note

バックエンド管理者実装は、管理者認証フローを使用しているため、フローは記憶されたデバイスをサポートしていません。デバイス追跡が有効にしている場合、管理者認証は成功しますが、アクセストークンの更新に関する呼び出しはいずれも失敗します。

クライアント側の認証オプション

モバイルアプリやその他のクライアント側のアプリケーションタイプは、ユーザーのデバイスにインストールされ、認証とユーザーインターフェイスのロジックをローカルで実行します。一般的に、以下の特性があります。

- React native、Flutter、Swift などの言語で構築され、ユーザーデバイスにデプロイされます。
- パブリッククライアントと呼ばれる、クライアントシークレットを持たないユーザープールの[アプリケーションクライアント](#)に接続します。
- IAM 認可 API リクエストを承認する AWS 認証情報にアクセスできません。
- 認証のために[マネージドログイン](#)を呼び出すか、AWS SDK を使用してユーザープール API でパブリックオペレーションとトークン認可オペレーションを使用します。
- 一般のユーザーにサービスを提供しており、誰でもサインアップしてサインインできます。

ユーザープール API を使用したクライアント側のオペレーションでは、パスワード、ワンタイムパスワード、またはパスキーを主要なサインイン要素として使用できます。[AWS Amplify](#) または [AWS SDK](#) で作成したユーザークライアント側のアプリケーションの場合、以下の手順で動作します。

1. ユーザーがアプリにユーザー名とパスワードを入力します。
2. アプリケーションが、ユーザーのユーザー名と SRP (Secure Remote Password) 詳細を使用して、InitiateAuth オペレーションを呼び出します。

この API オペレーションは、認証のパラメータを返します。

Note

アプリは、AWS SDK に組み込まれている Amazon Cognito SRP 機能を使用して SRP の詳細を生成します。

3. アプリが RespondToAuthChallenge 操作を呼び出します。呼び出しが成功すると、Amazon Cognito からユーザーのトークンが返され、認証フローが完了します。

Amazon Cognito に必要なチャレンジが別に存在する場合は、RespondToAuthChallenge を呼び出してもトークンは返されません。代わりに、呼び出しによってセッションが返されます。

4. RespondToAuthChallenge からセッションが返された場合、アプリは RespondToAuthChallenge を再度呼び出します。今回の呼び出しには、セッションとチャレンジのレスポンス (MFA コードなど) が使用されます。

API、OIDC、マネージドログインページの認証についての理解

Amazon Cognito ユーザープールは、いくつかの認証テクノロジーの組み合わせです。ユーザープールは、外部 ID プロバイダー (IdP) に依拠しているパーティであり、OpenID Connect (OIDC) SDK を使用して認証を実装するアプリケーションの IdP です。AWS SDK JSON ウェブトークン (JWT) の発行者として、OIDC 認証に類似した認証を (ただし、SDK の一部である API メソッドで) 提供します。また、アプリケーションへの安全なエントリポイントにもなります。

サインアップ、サインイン、ユーザープール内のユーザー管理を行うには、2 つのオプションがあります。

1. マネージドログインページとクラシックのホストされた UI には、[マネージドログインのユーザーインタラクティブエンドポイント](#)と、IdP および依拠しているパーティのロールを処理する[フェデレーションエンドポイント](#)が含まれています。これらは、ユーザープールの[ドメインを選択すると](#) Amazon Cognito がアクティブ化するパブリックウェブページのパッケージを構成します。サインアップ、サインイン、パスワード管理、多要素認証 (MFA) のページなど、Amazon Cognito ユーザープールの認証および認可機能をすぐに使い始めるには、マネージドログインの組み込みユーザーインターフェイスを使用します。

その他のユーザープールエンドポイントを使うと、サードパーティーの ID プロバイダー (IdP) を使った認証が容易になります。行われるサービスには以下が含まれます。

- a. IdPs からの認証済みクレーム用のサービスプロバイダーのコールバックエンドポイント (saml2/idpresponse や oauth2/idpresponse など)。Amazon Cognito がアプリと IdP の間の中間サービスプロバイダー (SP) である場合、コールバックエンドポイントはサービスを表します。
 - b. 環境に関する情報を提供するエンドポイント (oauth2/userInfo や /.well-known/jwks.json など)。アプリケーションは、OIDC または OAuth 2.0 デベロッパーライブラリを使用してトークンを検証したり、ユーザープロフィールデータを取得したりするときに、これらのエンドポイントを使用します。
2. [Amazon Cognito ユーザープール API](#) は、ウェブアプリやモバイルアプリが独自のカスタムフロントエンドでサインイン情報を収集した後で、ユーザーを認証する一連のツールです。ユーザープール API 認証では、次の JSON ウェブトークンが生成されます。
 - a. ユーザーからの検証可能な属性クレームを含む ID トークン。
 - b. [AWS サービスエンドポイント](#) に対する、トークンで認可された API リクエストを作成することをユーザーに許可するアクセストークン。

Note

デフォルトでは、ユーザープール API 認証のアクセストークンには `aws.cognito.signin.user.admin` スコープのみが含まれます。サードパーティー API へのリクエストを認可するなど、スコープを追加したアクセストークンを生成するには、ユーザープールエンドポイントで認証中にスコープをリクエストするか、[トークン生成前の Lambda トリガー](#)でカスタムスコープを追加します。アクセストークンのカスタマイズにより、AWS 請求書にコストがかかります。

- c. 新しい ID トークンとアクセストークンのリクエストを承認し、ユーザー ID とアクセスコントロールのプロパティを更新する更新トークン。

通常はユーザープールのエンドポイントを使用してサインインするフェデレーションユーザーを、プロファイルがユーザープールに対してローカルであるユーザーとリンクできます。ローカルユーザーは、外部 IdP を介したフェデレーションなしに、ユーザープールディレクトリにのみ存在します。[AdminLinkProviderForUser](#) API リクエストでフェデレーティッド ID をローカルユーザーにリンクすると、そのユーザーはユーザープール API でサインインできます。詳細については、「[フェデレーションユーザーを既存のユーザープロファイルにリンクする](#)」を参照してください。

Amazon Cognito ユーザープール API には 2 つの用途があります。

1. Amazon Cognito ユーザープールリソースを作成して設定します。たとえば、ユーザープールの作成、AWS Lambda トリガーの追加、マネージドログインページをホストするユーザープールドメインの設定を行うことができます。
2. ローカルユーザーおよびリンクされたユーザーのサインアップ、サインイン、その他のユーザーオペレーションを実行します。

Amazon Cognito ユーザープール API を使用したシナリオ例

1. ユーザーは、アプリで作成した [Create an account] (アカウントを作成) ボタンを選択します。E メールアドレスとパスワードを入力します。
2. アプリケーションは [SignUp](#) API リクエストを送信し、ユーザープールに新しいユーザーを作成します。
3. アプリケーションは、ユーザーに E メールの確認コードを求めます。ユーザーは、E メールメッセージで受け取ったコードを入力します。
4. アプリは、ユーザーの確認コードを含む [ConfirmSignUp](#) API リクエストを送信します。

5. アプリケーションは、ユーザーにユーザー名とパスワードの入力を求め、情報を入力します。
6. アプリは [InitiateAuth](#) API リクエストを送信し、ID トークン、アクセストークン、更新トークンを保存します。アプリは OIDC ライブラリを呼び出して、ユーザーのトークンを管理し、そのユーザーの永続セッションを維持します。

Amazon Cognito ユーザープール API では、IdP を介してフェデレートするユーザーをサインインすることはできません。これらのユーザーは、ユーザープールのエンドポイントを使用して認証する必要があります。マネージドログインを含むユーザープールエンドポイントの詳細については、「[ユーザープールのエンドポイントとマネージドログインのリファレンス](#)」を参照してください。

フェデレーションユーザーは、マネージドログインから開始して IdP を選択することも、マネージドログインをスキップしてユーザーを IdP に直接送信してサインインさせることもできます。[認可エンドポイント](#) への API リクエストに IdP パラメータが含まれている場合、Amazon Cognito はユーザーを IdP サインインページにそのままリダイレクトします。

マネージドログインページを使用したシナリオ例

1. ユーザーは、アプリで作成した [Create an account] (アカウントを作成) ボタンを選択します。
2. マネージドログインでは、デベロッパーの認証情報を登録したソーシャル ID プロバイダーのリストをユーザーに提示します。ユーザーは Apple を選択します。
3. アプリは、プロバイダー名 `SignInWithApple` で [認可エンドポイント](#) へのリクエストを開始します。
4. ユーザーのブラウザで Apple 認証ページが開きます。ユーザーはサインインし、Amazon Cognito にプロフィール情報の読み取りを許可することを選択します。
5. Amazon Cognito は Apple アクセストークンを確認し、ユーザーの Apple プロファイルを照会します。
6. ユーザーは Amazon Cognito 認可コードをアプリケーションに提示します。
7. アプリケーション内の OIDC ライブラリは、認証コードを [トークンエンドポイント](#) と交換し、ユーザープールから発行された ID トークン、アクセストークン、更新トークンを保存します。アプリケーションは、OIDC ライブラリを使用してユーザーのトークンを管理し、ユーザーの永続的なセッションを維持します。

ユーザープール API とマネージドログインページは、このガイド全体で説明しているさまざまなシナリオをサポートしています。次のセクションでは、ユーザープール API がサインアップ、サインイン、およびリソース管理の要件をサポートするクラスにさらにどのように分類されるかを見ていきます。

認可モデル別にグループ化された API オペレーションのリスト

Amazon Cognito ユーザープール API は、リソース管理インターフェイスとユーザー向け認証および認可インターフェイスの両方であり、運用に続く承認モデルを組み合わせます。API オペレーションによっては、IAM 認証情報、アクセストークン、セッショントークン、クライアントシークレット、またはこれらの組み合わせを使用して認可を行う必要がある場合があります。多くのユーザー認証および認可操作では、リクエストの認証済みバージョンと認証されていないバージョンを選択できます。認証されていない操作は、モバイルアプリなど、ユーザーに配布するアプリのセキュリティ上のベストプラクティスです。コードにシークレットを含める必要はありません。

IAM ポリシーでは、[IAM で認可された管理オペレーション](#) と [IAM で認可されたユーザーオペレーション](#) にのみアクセス許可を割り当てることができます。

IAM で認可された管理オペレーション

IAM で認可された管理オペレーションでは、AWS マネジメントコンソールで行う場合と同様に、ユーザープールとアプリケーションクライアントの設定を変更および表示します。

例えば、[UpdateUserPool](#) API リクエストでユーザープールを変更するには、リソースを更新するための AWS 認証情報と IAM アクセス許可を提示する必要があります。

AWS Command Line Interface (AWS CLI) または AWS SDK でこれらのリクエストを承認するには、リクエストに IAM 認証情報を追加する環境変数またはクライアント設定を使用して環境を設定します。詳細については、『』の[AWS 「認証情報 AWS を使用した へのアクセス」](#)を参照してください。AWS 全般のリファレンス。Amazon Cognito ユーザープール API の[サービスエンドポイント](#)にリクエストを直接送信することもできます。リクエストのヘッダーに埋め込む AWS 認証情報を使用して、これらのリクエストを承認または署名する必要があります。詳細については、[AWS 「API リクエストの署名」](#)を参照してください。

IAM で認可された管理オペレーション

[AddCustomAttributes](#)

[CreateGroup](#)

[CreateIdentityProvider](#)

[CreateResourceServer](#)

[CreateUserImportJob](#)

IAM で認可された管理オペレーション

[CreateUserPool](#)

[CreateUserPoolClient](#)

[CreateUserPoolDomain](#)

[DeleteGroup](#)

[DeleteIdentityProvider](#)

[DeleteResourceServer](#)

[DeleteUserPool](#)

[DeleteUserPoolClient](#)

[DeleteUserPoolDomain](#)

[DescribeIdentityProvider](#)

[DescribeResourceServer](#)

[DescribeRiskConfiguration](#)

[DescribeUserImportJob](#)

[DescribeUserPool](#)

[DescribeUserPoolClient](#)

[DescribeUserPoolDomain](#)

[GetCSVHeader](#)

[GetGroup](#)

[GetIdentityProviderByIdentifier](#)

[GetSigningCertificate](#)

[GetUICustomization](#)

IAM で認可された管理オペレーション

[GetUserPoolMfaConfig](#)

[ListGroups](#)

[ListIdentityProviders](#)

[ListResourceServers](#)

[ListTagsForResource](#)

[ListUserImportJobs](#)

[ListUserPoolClients](#)

[ListUserPools](#)

[ListUsers](#)

[ListUsersInGroup](#)

[SetRiskConfiguration](#)

[SetUICustomization](#)

[SetUserPoolMfaConfig](#)

[StartUserImportJob](#)

[StopUserImportJob](#)

[TagResource](#)

[UntagResource](#)

[UpdateGroup](#)

[UpdateIdentityProvider](#)

[UpdateResourceServer](#)

[UpdateUserPool](#)

IAM で認可された管理オペレーション

[UpdateUserPoolClient](#)

[UpdateUserPoolDomain](#)

IAM で認可されたユーザーオペレーション

IAM で認可されたユーザーオペレーションは、ユーザーのサインアップ、サインイン、認証情報の管理、変更、表示を行います。

例えば、ウェブフロントエンドをバックアップするサーバー側のアプリケーション層を設定できます。これは、Amazon Cognito リソースへの特権アクセスで信頼する OAuth 機密クライアントです。ユーザーをアプリに登録するには、サーバーで [AdminCreateUser](#) API リクエストに AWS 認証情報を含めることができます。OAuth クライアントタイプの詳細については、「The OAuth 2.0 Authorization Framework」の「[Client Types](#)」を参照してください。

AWS CLI または AWS SDK でこれらのリクエストを認可するには、環境変数またはリクエストに IAM 認証情報を追加するクライアント設定を使用してサーバー側のアプリケーション環境を設定します。詳細については、『』の[AWS「認証情報 AWS を使用した へのアクセス」](#)を参照してください。AWS 全般のリファレンス。Amazon Cognito ユーザープール API の[サービスエンドポイント](#)にリクエストを直接送信することもできます。リクエストのヘッダーに埋め込む AWS 認証情報を使用して、これらのリクエストを承認または署名する必要があります。詳細については、[AWS「API リクエストの署名」](#)を参照してください。

アプリケーションクライアントにクライアントシークレットがある場合は、IAM 認証情報と、オペレーションによっては、AuthParameters で SecretHash パラメータまたは SECRET_HASH 値の両方を指定する必要があります。詳細については、「[シークレットハッシュ 値の計算](#)」を参照してください。

IAM で認可されたユーザーオペレーション

[AdminAddUserToGroup](#)

[AdminConfirmSignUp](#)

[AdminCreateUser](#)

[AdminDeleteUser](#)

IAM で認可されたユーザーオペレーション

[AdminDeleteUserAttributes](#)

[AdminDisableProviderForUser](#)

[AdminDisableUser](#)

[AdminEnableUser](#)

[AdminForgetDevice](#)

[AdminGetDevice](#)

[AdminGetUser](#)

[AdminInitiateAuth](#)

[AdminLinkProviderForUser](#)

[AdminListDevices](#)

[AdminListGroupsWithUser](#)

[AdminListUserAuthEvents](#)

[AdminRemoveUserFromGroup](#)

[AdminResetUserPassword](#)

[AdminRespondToAuthChallenge](#)

[AdminSetUserMFAPreference](#)

[AdminSetUserPassword](#)

[AdminSetUserSettings](#)

[AdminUpdateAuthEventFeedback](#)

[AdminUpdateDeviceStatus](#)

[AdminUpdateUserAttributes](#)

IAM で認可されたユーザーオペレーション

[AdminUserGlobalSignOut](#)

認証されていないユーザーオペレーション

認証されていないユーザーオペレーションでは、ユーザーのサインアップ、サインイン、およびパスワードのリセットが開始されます。インターネット上の誰でもがアプリケーションにサインアップしてサインインできるようにする場合は、認証されていない API オペレーション、またはパブリック API オペレーションを使用します。

例えば、アプリケーションにユーザーを登録するには、シークレットへの特権アクセスを提供しない OAuth パブリッククライアントを配布できます。このユーザーは、認証されていない API オペレーション [SignUp](#) で登録できます。

AWS SDK で開発したパブリッククライアントでこれらのリクエストを送信するには、認証情報を設定する必要はありません。また、Amazon Cognito ユーザープール API の [サービスエンドポイント](#) に、追加の認可なしでリクエストを直接送信することもできます。

アプリケーションクライアントにクライアントシークレットがある場合は、オペレーションによっては、AuthParameters で SecretHash パラメータまたは SECRET_HASH 値を指定する必要があります。詳細については、「[シークレットハッシュ 値の計算](#)」を参照してください。

認証されていないユーザーオペレーション

[SignUp](#)

[ConfirmSignUp](#)

[ResendConfirmationCode](#)

[ForgotPassword](#)

[ConfirmForgotPassword](#)

[InitiateAuth](#)

トークン認証によるユーザーオペレーション

トークン認証によるユーザーオペレーションは、ユーザーがサインインしたか、サインインプロセスを開始した後に、ユーザーのサインアウト、認証情報の管理、変更、表示を行います。アプリケーション内でシークレットを配布しない場合や、ユーザー自身の認証情報でリクエストを認可する場合は、トークン認可による API オペレーションを使用してください。ユーザーがサインインを完了した場合、ユーザーのトークン認可された API リクエストをアクセストークンで認可する必要があります。ユーザーがサインインプロセス中である場合は、Amazon Cognito が前のリクエストへのレスポンスで返したセッショントークンを使用して、トークン認可による API リクエストを認可する必要があります。

例えば、パブリッククライアントでは、書き込みアクセスをユーザー自身のプロフィールのみに制限する方法でユーザーのプロフィールを更新する場合があります。この更新を行うには、クライアントが [UpdateUserAttributes](#) API リクエストにユーザーのアクセストークンを含めることができます。

AWS SDK で開発したパブリッククライアントでこれらのリクエストを送信するには、認証情報を設定する必要はありません。リクエストには AccessToken または Session パラメータを含めてください。Amazon Cognito ユーザープール API の [サービスエンドポイント](#) にリクエストを直接送信することもできます。サービスエンドポイントへのリクエストを承認するには、リクエストの POST 本文にアクセストークンまたはセッショントークンを含めます。

トークン認証オペレーションの API リクエストに署名するには、アクセストークンをリクエストの Authorization ヘッダーとして Bearer *<Base64-encoded access token>* 形式で含めます。

トークン認証による ユーザーオペレーション	AccessTok en	Session

RespondToAuthChallenge		✓

ChangePassword	✓	

GetUser	✓	

StartWebAuthnRegistration	✓	

トークン認証による ユーザーオペレーション	AccessTok en	Session
CompleteWebAuthnRegistration	✓	
DeleteWebAuthnCredential	✓	
ListWebAuthnCredent tials	✓	
UpdateUserAttributes	✓	
DeleteUserAttributes	✓	
DeleteUser	✓	
ConfirmDevice	✓	
ForgetDevice	✓	
GetDevice	✓	
ListDevices	✓	
UpdateDeviceStatus	✓	
GetUserAttributeVe rificationCode	✓	
VerifyUserAttribute	✓	
SetUserSettings	✓	
SetUserMFAPreferen ce	✓	
GlobalSignOut	✓	

トークン認証による ユーザーオペレーション	AccessTok en	Session

UpdateAuthEventFeedback		✓

AssociateSoftwareToken	✓	✓

VerifySoftwareToken	✓	✓

[RevokeToken](#)¹

[GetTokensFromRefreshToken](#)¹

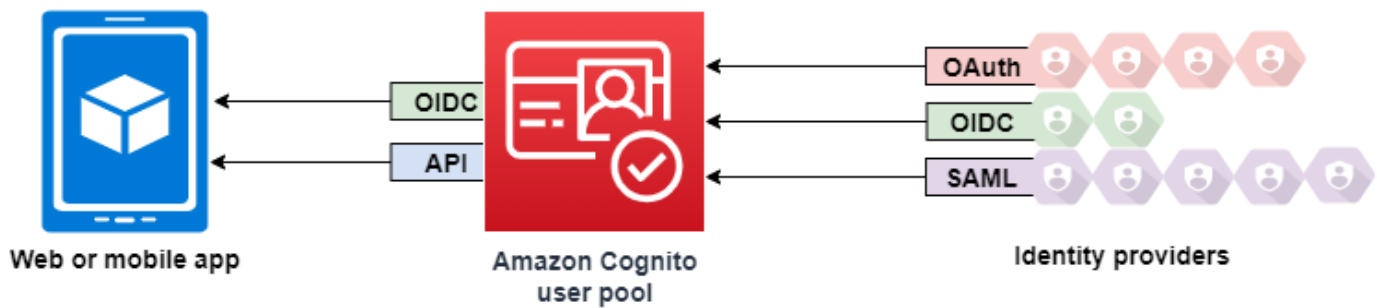
¹ RevokeToken と GetTokensFromRefreshToken は、更新トークンを認証パラメータとして受け取ります。更新トークンは、認証トークンとして、またターゲットリソースとして機能します。

サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン

アプリケーションユーザーは、ユーザープール経由で直接サインインするか、またはサードパーティーの ID プロバイダー (IdP) 経由でフェデレーション方式で認証を行うことができます。ユーザープールは、Facebook、Google、Amazon、Apple 経由のソーシャルサインイン、および OpenID Connect (OIDC) と SAML IdP から返されるトークンの処理のオーバーヘッドを管理します。組み込みの Hosted Web UI では、Amazon Cognito がすべての認証済みユーザーに関するトークンの処理と管理を提供します。このように、バックエンドシステムは 1 セットのユーザープールトークンで標準化できます。

Amazon Cognito ユーザープールでのフェデレーションサインインの仕組み

サードパーティー (フェデレーション) 経由のサインインは、Amazon Cognito のユーザープールで使用できます。この機能は、Amazon Cognito ID プール (フェデレーテッド ID) 経由のフェデレーションとは無関係です。



Amazon Cognito はユーザーディレクトリであり、OAuth 2.0 ID プロバイダー (IdP) です。Amazon Cognito ディレクトリにローカルユーザーをサインインさせると、ユーザープールがアプリの IdP になります。ローカルユーザーは、外部 IdP を介したフェデレーションなしに、ユーザープールディレクトリにのみ存在します。

Amazon Cognito をソーシャル、SAML、または OpenID Connect (OIDC) IdP に接続すると、ユーザープールは複数のサービスプロバイダーとアプリ間のブリッジとして機能します。IdP によって、Amazon Cognito はサービスプロバイダー (SP) です。IdP は、OIDC ID トークンまたは SAML アサーションを Amazon Cognito に渡します。Amazon Cognito は、トークンまたはアサーション内のユーザーに関するクレームを読み取り、それらのクレームをユーザープールディレクトリ内の新しいユーザープロフィールにマッピングします。

Amazon Cognito は、フェデレーションユーザーのユーザープロフィールを独自のディレクトリに作成します。Amazon Cognito は、IdP からのクレームと、OIDC およびソーシャル ID プロバイダーの場合は、IdP によるパブリック `userinfo` エンドポイントに基づいて、ユーザーに属性を追加します。マッピングされた IdP 属性が変更されると、ユーザーの属性はユーザープール内で変化します。さらに、IdP の属性から独立した属性を追加することもできます。

Amazon Cognito がフェデレーションユーザーのプロファイルを作成すると、その機能が変更され、SP となったアプリに IdP として表示されます。Amazon Cognito は OIDC と OAuth 2.0 IdP の組み合わせです。アクセストークン、ID トークン、およびリフレッシュトークンを生成します。トークンの詳細については、[ユーザープール JSON ウェブトークン \(JWT\) の理解](#) を参照してください。

フェデレーションまたはローカルを問わず、ユーザーを認証して認可するために、Amazon Cognito と統合するアプリを設計する必要があります。

Amazon Cognito のサービスプロバイダーとしてのアプリの責任

トークンの情報を検証して処理する

ほとんどのシナリオでは、Amazon Cognito は認証されたユーザーを、認可コードを追加したアプリの URL にリダイレクトします。アプリは、アクセス、ID、およびリフレッシュの各トークンと[このコードを交換](#)します。そして、[トークンの有効性を確認](#)し、トークン内のクレームに基づいて、ユーザーに情報を提供する必要があります。

Amazon Cognito API リクエストによる認証イベントへの応答

アプリは、[Amazon Cognito ユーザープール API](#) と [認証 API エンドポイント](#) と統合する必要があります。認証 API は、ユーザーのサインインとサインアウトを行い、トークンを管理します。ユーザープール API には、ユーザープール、ユーザー、および認証環境のセキュリティを管理するさまざまなオペレーションがあります。アプリは、Amazon Cognito からの応答を受信したときに次に何をすべきかを知っている必要があります。

Amazon Cognito ユーザープールサードパーティーサインインについて知っておくべきこと

- フェデレーションプロバイダーでユーザーにサインインさせたい場合は、ドメインを選択する必要があります。これにより、[マネージドログイン](#)用のページが設定されます。詳細については、「[マネージドログインに独自のドメインを使用する](#)」を参照してください。
- [InitiateAuth](#) や [AdminInitiateAuth](#) のような API オペレーションを使用してフェデレーションユーザーでサインインすることはできません。フェデレーションユーザーは、[ログインエンドポイント](#) または [認可エンドポイント](#) でのみサインインできます。
- [認可エンドポイント](#) はリダイレクションエンドポイントです。リクエストに `idp_identifier` パラメータまたは `identity_provider` パラメータを指定すると、マネージドログインをバイパスして、IdP にサイレントにリダイレクトされます。それ以外の場合は、マネージドログインの[ログインエンドポイント](#)にリダイレクトされます。
- マネージドログインがセッションをフェデレーティッド IdP にリダイレクトすると、Amazon Cognito は、Amazon/Cognito リクエストに `user-agent` ヘッダーを含めます。
- Amazon Cognito は、フェデレーションユーザープロファイルの `username` 属性を、固定識別子と IdP の名前の組み合わせから派生します。カスタム要件に一致するユーザー名を生成するには、`preferred_username` 属性へのマッピングを作成します。詳細については、「[マッピングについて知っておくべきこと](#)」を参照してください。

例: MyIDP_bob@example.com

- Amazon Cognito は、ユーザープールに追加する OIDC、SAML、ソーシャル IdP ごとに [ユーザーグループ](#) を作成します。グループ名の形式は [user pool ID]_[IdP name] です (例: us-east-1_EXAMPLE_MYSSO、us-east-1_EXAMPLE_Google)。自動生成された一意の各 IdP ユーザープロファイルは、このグループに自動的に追加されます。[リンクされたユーザー](#) は、このグループに自動的に追加されませんが、そのプロファイルを別のプロセスでグループに追加できます。
- Amazon Cognito は、フェデレーションユーザーの ID に関する情報を属性および identities と呼ばれる ID トークン内のクレームに記録します。このクレームには、ユーザーのプロバイダーと、プロバイダーからの一意の ID が含まれます。ユーザープロファイル内の identities 属性を直接変更することはできません。フェデレーションユーザーをリンクする方法の詳細については、「[フェデレーションユーザーを既存のユーザープロファイルにリンクする](#)」を参照してください。
- [UpdateIdentityProvider](#) API リクエストで IdP を更新する場合、変更がマネージドログインに反映されるまでに最大で 1 分かかることがあります。
- Amazon Cognito は、自身と IdP との間で最大 20 の HTTP リダイレクトをサポートします。
- ユーザーがマネージドログインを使用してサインインすると、ブラウザは、暗号化されたログインセッション Cookie を保存することで、サインインに使用したクライアントとプロバイダーを記録します。ユーザーが同じパラメータを使用して再度サインインしようとする、マネージドログインの有効期限が切れていない既存のセッションをすべて再利用するため、ユーザーは認証情報を再入力することなく認証されます。ローカルユーザープールログインへの切り替えやローカルユーザープールログインからの切り替えなど、ユーザーが別の IdP で再度ログインする場合は、認証情報を入力して新しいログインセッションを生成する必要があります。

ユーザープール IdPs は任意のアプリクライアントに割り当てることができ、ユーザーはアプリクライアントに割り当てた IdP でのみサインインできます。

トピック

- [ユーザープールの ID プロバイダーの設定](#)
- [ユーザープールによるソーシャル ID プロバイダーの使用](#)
- [ユーザープールによる SAML ID プロバイダーの使用](#)
- [ユーザープールでの OIDC ID プロバイダーの使用](#)
- [IdP 属性をプロファイルとトークンにマッピングする](#)

- [フェデレーションユーザーを既存のユーザープロフィールにリンクする](#)

ユーザープールの ID プロバイダーの設定

ユーザープールを使用すると、さまざまな外部 ID プロバイダー (IdP) を通じてサインインを実装できます。ガイドのこのセクションには、Amazon Cognito コンソールでユーザープールを使用してこれらの ID プロバイダーを設定する手順が記載されています。または、ユーザープール API と AWS SDK を使用して、ユーザープール ID プロバイダーをプログラムで追加することもできます。詳細については、「[CreateIdentityProvider](#)」を参照してください。

サポートされている ID プロバイダーのオプションには、Facebook、Google、Amazon などのソーシャルプロバイダー、OpenID Connect (OIDC)、SAML 2.0 プロバイダーなどがあります。開始する前に、IdP の管理認証情報を設定します。プロバイダーのタイプごとに、アプリケーションの登録、必要な認証情報の取得、ユーザープールでのプロバイダーの詳細の設定を行う必要があります。その後、ユーザーは、接続された ID プロバイダーの既存アカウントを使用して、アプリケーションへのサインアップとサインインができます。

[認証] の下にある [ソーシャルプロバイダーと外部プロバイダー] メニューで、ユーザープールの IdP を追加および更新します。詳細については、「[サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン](#)」を参照してください。

トピック

- [ソーシャル IdP でユーザーサインインを設定する](#)
- [OIDC IdP でユーザーサインインを設定する](#)
- [SAML IdP でユーザーサインインを設定する](#)

ソーシャル IdP でユーザーサインインを設定する

フェデレーションを使用して、Amazon Cognito のユーザープールを Facebook、Google、Login with Amazon などのソーシャル ID プロバイダーと統合することができます。

ソーシャル ID プロバイダーを追加するには、最初に ID プロバイダーでデベロッパーアカウントを作成します。デベロッパーアカウントが作成されたら、アプリを ID プロバイダーに登録します。ID プロバイダーがアプリ用のアプリ ID とアプリシークレットを作成するので、これらの値を Amazon Cognito ユーザープールで設定します。

- [Google Identity Platform](#)

- [Facebook for Developers](#)
- [Login with Amazon](#)
- [Apple でサインイン](#)

ユーザーサインインとソーシャル IdP を統合するには

1. [Amazon Cognito コンソール](#)にサインインします。プロンプトが表示されたら、AWS 認証情報を入力します。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択します。
4. [Add an identity provider] (ID プロバイダーを追加する) を選択、または、設定した [Facebook]、[Google]、[Amazon]、または [Apple] などを選択し、[Identity provider information] (ID プロバイダー情報) を検索し、[Edit] (編集) をクリックします。ソーシャル ID プロバイダーの追加の詳細については、「[ユーザープールによるソーシャル ID プロバイダーの使用](#)」を参照してください。
5. 選択した IdP に基づいて、次のいずれかの手順を実行して、ソーシャル ID プロバイダーの情報を入力します。

Facebook、Google、および Login with Amazon

クライアントアプリを作成したときに受け取ったアプリ ID とアプリシークレットを入力します。

Apple でサインイン

Apple に提供したサービス ID、およびアプリケーションクライアントを作成したときに受け取ったチーム ID、キー ID、プライベートキーを入力します。

6. [Authorize scopes] (承認スコープ) に、ユーザープール属性にマップするソーシャル ID プロバイダースコープの名前を入力します。スコープは、アプリケーションでアクセスするユーザー属性 (名前や E メールなど) を定義します。スコープを入力するときは、選択した IdP に基づいて、次のガイドラインに従ってください。

- Facebook — スコープはカンマで区切ります。例:

`public_profile, email`

- Google、Login with Amazon、Sign in with Apple — スコープをスペースで区切ります。例:

- Google: profile email openid
- Login with Amazon: profile postal_code
- Sign In with Apple: name email

 Note

Sign in with Apple (コンソール) の場合は、チェックボックスを使用してスコープを選択します。

7. [Save changes] (変更の保存) をクリックします。
8. [アプリケーションクライアント] メニューで、リストからアプリケーションクライアントを選択し、[編集] を選択します。[Identity providers] (ID プロバイダー) で、新しいソーシャル ID プロバイダーをアプリケーションクライアントに追加します。
9. [Save changes] (変更の保存) をクリックします。

ソーシャル IdP の詳細については、「[ユーザープールによるソーシャル ID プロバイダーの使用](#)」を参照してください。

OIDC IdP でユーザーサインインを設定する

Salesforce や Ping Identity などの OpenID Connect (OIDC) ID プロバイダーとユーザーサインインを統合することができます。

OIDC プロバイダーをユーザープールに追加するには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. ナビゲーションメニューから [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択し、[ID プロバイダーを追加] を選択します。
5. [OpenID Connect] (OpenID 接続) ID プロバイダーを選択します。
6. [Provider name] (プロバイダー名) に一意の名前を入力します。
7. プロバイダーから受け取ったクライアント ID を、[Client ID] (クライアント ID) へ入力します。
8. プロバイダーから受け取ったクライアントシークレットを、[Client secret] (クライアントシークレット) に入力します。

9. このプロバイダーの [Authorized scopes] (承認済みスコープ) を入力します。スコープは、アプリケーションがプロバイダーにリクエストするユーザー属性のグループ (name および email など) を定義します。[OAuth 2.0](#) 仕様に従い、スコープはスペースで区切る必要があります。

ユーザーはこれらの属性をアプリケーションに提供することに同意する必要があります。

10. [Attribute request method] (属性リクエストメソッド) を選択して、プロバイダーが操作する [userInfo] エンドポイントからユーザーの詳細をフェッチするために必要な HTTP メソッド (GET または POST) を Amazon Cognito に提供します。
11. [Setup method] (セットアップ方法) を選択して、OpenID Connect エンドポイントを、[Auto fill through issuer URL] (発行者 URL による自動入力) または [Manual input] (手動入力) で取得します。[発行者 URL を通じた自動入力] は、プロバイダーにパブリック .well-known/openid-configuration エンドポイントがあり、これを通じて Amazon Cognito が authorization、token、userInfo、jwks_uri エンドポイントの URL を取得できる場合に使用します。
12. 発行者の URL、または IdP からの authorization、token、userInfo、および jwks_uri エンドポイントの URL を入力します。

Note

検出、自動入力、および手動で入力された URL には、ポート番号 443 と 80 のみを使用できます。OIDC プロバイダーが標準外の TCP ポートを使用している場合、ユーザーログインは失敗します。

発行者の URL は https:// で始まる必要があり、/ 文字で終わらせることはできません。例えば、Salesforce では次の URL を使用します。

`https://login.salesforce.com`

発行者 URL に関連付けられている openid-configuration ド

キュメントには、次の値の HTTPS URL を指定する必要があります:

authorization_endpoint、token_endpoint、userinfo_endpoint、および jwks_uri。同様に、[Manual input] (手動入力) を選択する場合は、HTTPS URL のみを入力できます。

13. デフォルトで、OIDC クレームの [sub] (サブ) はユーザープール属性の [Username] (ユーザーネーム) にマッピングされます。他の OIDC [クレーム](#) をユーザープール属性にマッピングできます。OIDC クレームを入力し、対応するユーザープール属性をドロップダウンリストから選択します。例えば、通常、クレームの [email] はユーザープール属性の [E メール] にマッピングされます。

14. ID プロバイダーからユーザープールに追加の属性をマッピングします。詳細は、「[ユーザープール用 ID プロバイダー属性マッピングの特定](#)」を参照してください。
15. [作成] を選択します。
16. [アプリケーションクライアント] メニューで、リストからアプリケーションクライアントを選択します。新しい SAML ID プロバイダーをアプリケーションクライアントに追加するには、[ログインページ] タブに移動し、[マネージドログインページ設定] の [編集] を選択します。
17. [Save changes] (変更の保存) をクリックします。

OIDC IdPs の詳細については、「[ユーザープールでの OIDC ID プロバイダーの使用](#)」を参照してください。

SAML IdP でユーザーサインインを設定する

Amazon Cognito ユーザープールのフェデレーションを使用して、SAML ID プロバイダー (IdP) と統合することができます。メタデータドキュメントを指定します。ファイルをアップロードするか、メタデータドキュメントのエンドポイント URL を入力します。サードパーティーの SAML IdP のメタデータドキュメントの取得については、「[サードパーティー SAML ID プロバイダーの設定](#)」を参照してください。

ユーザープールに SAML 2.0 ID プロバイダーを設定する

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択し、[ID プロバイダーを追加] を選択します。
5. [SAML] ID プロバイダーを選択します。
6. カンマで区切られた [Identifiers] (識別子) を入力します。識別子は Amazon Cognito に、ユーザーのサインイン E メールアドレスを確認し、ドメインに対応するプロバイダーに誘導する必要があることを伝えます。
7. ユーザーがログアウトしたときに、Amazon Cognito が署名されたサインアウト要求をプロバイダーに送信するためには、[Add sign-out flow] (サインアウトフローの追加) を選択します。マネージドログインの設定時に Amazon Cognito で作成した `https://mydomain.auth.us-east-1.amazoncognito.com/saml2/logout` エンドポイントにサインアウトレスポンスを

送信するように、SAML 2.0 ID プロバイダーを設定します。saml2/logout エンドポイントでは、ポストバインディングを使用します。

 Note

このオプションを選択し、SAML ID プロバイダーが署名付きログアウトリクエストを期待する場合は、SAML IdP で Amazon Cognito が提供する署名証明書を設定する必要があります。

SAML IdP は署名されたログアウトリクエストを処理し、Amazon Cognito セッションからユーザーをログアウトさせます。

8. [Metadata document source] (メタデータドキュメントソース) を選択します。ID プロバイダーがパブリック URL で SAML メタデータを提供する場合は、[Metadata document URL] (メタデータドキュメント URL) を選択してそのパブリック URL を入力できます。それ以外の場合は、[Upload metadata document] (メタデータドキュメントをアップロード) を選択し、プロバイダーから以前ダウンロードしたメタデータファイルを選択します。

 Note

プロバイダーにパブリックエンドポイントがある場合は、ファイルをアップロードするのではなく、メタデータドキュメントの URL を入力することをお勧めします。URL を使用する場合、Amazon Cognito はメタデータを自動的に更新します。通常、メタデータの更新は 6 時間ごとまたはメタデータの有効期限が切れる前のいずれか早いタイミングで発生します。

9. [Map attributes between your SAML provider and your app] (SAML プロバイダーとアプリ間で属性をマッピング) をクリックして、SAML プロバイダーの属性をユーザープールのユーザープロフィールにマッピングします。ユーザープールの必須属性を属性マップに含めます。

たとえば、[User pool attribute] (ユーザープール属性) email を選択する場合、ID プロバイダーからの SAML アサーションに表示される SAML 属性名を入力します。ID プロバイダーは、参考として SAML アサーションのサンプルを提供する場合があります。ID プロバイダーの中には、email などの単純な名前を使用するものもあれば、次のような URL 形式の属性名を使用するものもあります。

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

10. [作成] を選択します。

 Note

HTTPS メタデータエンドポイント URL を使用して SAML IdP を作成中に `InvalidParameterException` が表示される場合、メタデータエンドポイントの SSL が正しくセットアップされていること、および有効な SSL 証明書が関連付けられていることを確認してください。このような例外の例として、「Error retrieving metadata from `<metadata endpoint>`」が挙げられます。

署名証明書を追加するために SAML IdP をセットアップする

- IdP が署名付きログアウトリクエストの検証に使用するパブリックキーを含む証明書を取得するには、次の手順を実行します。
 1. ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューに移動します。
 2. SAML プロバイダーを選択します。
 3. [署名証明書を表示] を選択します。

SAML IdP の詳細については、「[ユーザープールによる SAML ID プロバイダーの使用](#)」を参照してください。

ユーザープールによるソーシャル ID プロバイダーの使用

ウェブおよびモバイルアプリのユーザーは、Facebook、Google、Amazon、Apple などのソーシャル ID プロバイダー (IdP) 経由でサインインできます。組み込みの Hosted Web UI では、Amazon Cognito がすべての認証済みユーザーに関するトークンの処理と管理を提供します。このように、バックエンドシステムは 1 セットのユーザープールトークンで標準化できます。サポートされているソーシャル ID プロバイダーと統合するには、マネージドログインを有効にする必要があります。Amazon Cognito は、マネージドログインページの構築時に OAuth 2.0 エンドポイントを作成します。Amazon Cognito、OIDC IdP、ソーシャル IdP は、このエンドポイントを使用して情報を交換します。詳細については、[Amazon Cognito API リファレンス](#)を参照してください。

ソーシャル IdP を に追加するか AWS マネジメントコンソール、CLI または Amazon Cognito API AWS を使用できます。

Note

サードパーティー (フェデレーション) 経由のサインインは、Amazon Cognito のユーザープールで使えます。この機能は、Amazon Cognito ID プール (フェデレーテッド ID) 経由のフェデレーションとは無関係です。

トピック

- [ソーシャル IdP デベロッパーのアカウントとアプリケーションを設定する](#)
- [ソーシャル IdP を使用してユーザープールを設定する](#)
- [ソーシャル IdP の設定をテストする](#)

ソーシャル IdP デベロッパーのアカウントとアプリケーションを設定する

Amazon Cognito でソーシャル IdP を作成する前に、アプリケーションをソーシャル IdP に登録して、クライアント ID とクライアントシークレットを取得する必要があります。

Facebook

Meta デベロッパーのアカウントと認証の設定に関する最新情報については、「[Meta でのアプリ開発](#)」を参照してください。

アプリケーションを Facebook/Meta に登録する方法

1. [Facebook の開発者アカウント](#)を作成します。
2. Facebook 認証情報を使用して[サインイン](#)します。
3. [My Apps] (マイアプリ) メニューから、[新しいアプリを作成] (新しいアプリを作成) を選択します。
4. Facebook アプリケーションの名前を入力し、[Create App ID] (アプリケーション ID の作成) を選択します。
5. 左のナビゲーションバーで、[設定]、[ベーシック] の順に選択します。
6. [App ID] (アプリ ID) と [App Secret (アプリシークレット)] を書き留めます。これらは次のセクションで使用します。
7. ページの下部で、[+ プラットフォームを追加] を選択します。
8. [Website] (ウェブサイト) を選択します。

9. [Website] (ウェブサイト) で、アプリケーションのサインインページへのパスを [Site URL] (サイト URL) に入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/login?
response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com
```

10. [Save changes] (変更の保存) をクリックします。
11. ユーザープールドメインのルートへのパスを [App Domains] (アプリケーションドメイン) に入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com
```

12. [Save changes] (変更の保存) をクリックします。
13. ナビゲーションバーで [Add Product] (プロダクトの追加) を選択し、[Facebook Login] (Facebook ログイン) の [Set up] (設定) を選択します。
14. ナビゲーションバーで [Facebook ログイン]、[設定] の順に選択します。

ユーザープールドメインの /oauth2/idpresponse エンドポイントへのパスを [Valid OAuth Redirect URIs] (有効な OAuth リダイレクト URI) に入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
```

15. [Save changes] (変更の保存) をクリックします。

Login with Amazon

Login with Amazon デベロッパーのアカウントと認証の設定に関する最新情報については、[「Login with Amazon に関するドキュメント」](#)を参照してください。

アプリケーションを Login with Amazon に登録する方法

1. [Amazon の開発者アカウント](#)を作成します。
2. Amazon 認証情報を使用して[サインイン](#)します。
3. Amazon クライアント ID およびクライアントシークレットを受け取るには、Amazon セキュリティプロファイルを作成する必要があります。

ページの上にあるナビゲーションバーで [Apps and Services (アプリとサービス)] を選択し、[Login with Amazon] を選択します。

4. [Create a New Security Profile (新しいセキュリティプロファイルの作成)] を選択します。
5. [Security Profile Name] (セキュリティプロファイル名)、[Security Profile Description] (セキュリティプロファイルの説明)、[Consent Privacy Notice URL] (プライバシー規約 URL の同意) に入力します。
6. [保存] を選択します。
7. [クライアント ID] および [クライアントシークレット] を選択して、クライアント ID およびシークレットを表示します。これらは次のセクションで使用します。
8. 歯車アイコンにマウスカーソルを合わせ、[Web Settings] (ウェブ設定)、[Edit] (編集) の順に選択します。
9. [Allowed Origins] (許可されたオリジン) にユーザープールのドメインを入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com
```

10. /oauth2/idpresponse エンドポイントを使用するユーザープールドメインを [Allowed Return URLs] (許可されたリターン URL) に入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
```

11. [保存] を選択します。

Google

Google Cloud プラットフォームの OAuth 2.0 の詳細については、「Google Workspace デベロッパーガイド」ドキュメントの「[認証と認可の詳細](#)」を参照してください。

アプリケーションを Google に登録する方法

1. [Google の開発者アカウント](#)を作成します。
2. [Google Cloud Platform コンソール](#)にサインインします。
3. 上部のナビゲーションバーから、[Select a project] (プロジェクトの選択) を選択します。Google プラットフォームにプロジェクトが既にある場合は、このメニューには代わりにデフォルトのプロジェクトが表示されます。
4. [NEW PROJECT] (新しいプロジェクト) を選択します。

5. 製品の名前を入力し、[CREATE] (作成) を選択します。
6. 左のナビゲーションバーで、[APIs and Services] (API とサービス) を選択し、[OAuth consent screen] (OAuth 同意画面) を選択します。
7. アプリ情報、[App domain] (アプリドメイン)、[Authorized domains] (承認済みドメイン)、[Developer contact information] (開発者の連絡先情報) を入力します。[Authorized domains] (承認済みドメイン) には、amazoncognito.com とカスタムドメインのルート (例: example.com) を含める必要があります。[SAVE AND CONTINUE] (保存して続行) を選択します。
8. 1. [Scopes] (スコープ) で、[Add or remove scopes] (スコープの追加または削除) を選択し、少なくとも、次の OAuth スコープを選択します。
 1. .../auth/userinfo.email
 2. .../auth/userinfo.profile
 3. openid
9. [Test users] (テストユーザー) で、[Add users] (ユーザーの追加) を選択します。E メールアドレスおよびその他の承認済みテストユーザーを入力して、[SAVE AND CONTINUE] (保存して続行) を選択します。
10. 左のナビゲーションバーを再度展開し、[APIs and Services] (API とサービス) を選択して、[Credentials] (認証情報) を選択します。
11. [CREATE CREDENTIALS] (認証情報の作成)、[OAuth client ID] (OAuth クライアント ID) の順に選択します。
12. [Application type] (アプリケーションタイプ) を選択し、クライアントに [Name] (名前) を入力します。
13. [Authorized JavaScript origins] (承認済みの JavaScript 生成元) で、[ADD URI] (URI の追加) を選択します。ユーザープールのドメインを入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com
```

14. [Authorized redirect URIs] (承認済みのリダイレクト URI) で、[ADD URI] (URI の追加) を選択します。ユーザープールのドメインの /oauth2/idpresponse エンドポイントへのパスを入力します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
```

15. [CREATE] (作成) を選択します。

16. [Your client ID] (クライアント ID) と [Your client secret] (クライアントシークレット) の下に Google が表示する値を安全に保存します。Google IdP を追加するときに、Amazon Cognito にこれらの値を指定します。

Sign in with Apple

「Apple でサインイン」の設定に関する最新の情報については、Apple デベロッパー向けドキュメントの「[Apple でサインインの環境を設定する](#)」を参照してください。

アプリケーションを Apple でサインイン (SIWA) に登録する方法

1. [Apple の開発者アカウント](#)を作成します。
2. Apple 認証情報を使用して[サインイン](#)します。
3. 左のナビゲーションバーで、[Certificates, Identifiers & Profiles] (証明書、ID & プロファイル) を選択します。
4. 左のナビゲーションペインで、[Identifiers] (識別子) を選択します。
5. [Identifiers] (識別子) ページで、[+] アイコンを選択します。
6. [Register a New Identifier] (新しい識別子の登録) ページで、[App IDs] (アプリ ID)、[Continue] (続行) の順に選択します。
7. [Select a type] (種類の選択) ページで、[App] (アプリ) を選択し、[Continue] (続行) を選択します。
8. [Register an App ID] ページで、次の操作を行います。
 1. [Description] (説明) で、説明を入力します。
 2. [App ID Prefix] (アプリ ID プレフィックス) に、[Bundle ID] (バンドル ID) を入力します。[App ID Prefix] (アプリケーション ID プレフィックス) にある値を書き留めておきます。この値は、[ソーシャル IDP を使用してユーザープールを設定する](#) で Apple を ID プロバイダーとして選択した後で使用します。
 3. [Capabilities] (機能) で、[Sign In with Apple] (Apple でサインイン) を選択してから [Edit] (編集) を選択します。
 4. [Sign in with Apple: App ID Configuration] (Apple でサインイン: アプリ ID の設定) ページで、アプリをプライマリとして設定するか、他のアプリ ID とグループ化するかを選択し、次に [Save] (保存) を選択します。
 5. [続行] をクリックしてください。
9. [Confirm your App ID] ページで、[登録] を選択します。

10. [Identifiers] (識別子) ページで、[+] アイコンを選択します。
11. [Register a New Identifier] (新しい識別子の登録) ページで、[Services IDs] (サービス ID)、[Continue] (続行) の順に選択します。
12. [Register a Services ID] ページで、次の操作を行います。
 1. [説明] に、説明を入力します。
 2. [Identifier] に、識別子を入力します。このサービス ID をメモしておきます。この値は、[ソーシャル IdP を使用してユーザープールを設定する](#) で Apple を ID プロバイダーとして選択した後で必要になります。
 3. [Continue] (続行) を選択し、[Register] (登録) を選択します。
13. [Identifiers] (識別子) ページから、先ほど作成した [Services ID] (サービス ID) を選択します。
 1. [Sign In with Apple] (Apple でサインイン) を選択後、[Configure] (設定) を選択します。
 2. [Web Authentication Configuration] (ウェブ認証の設定) ページで、[Primary App ID] (プライマリアプリ ID) として前に作成したアプリ ID を選択します。
 3. [Website URLs] (ウェブサイトの URL) の横の [+] アイコンを選択します。
 4. [Domains and subdomains] (ドメインとサブドメイン) で、https:// プレフィックスなしでユーザープールのドメインを入力します。

mydomain.auth.us-east-1.amazoncognito.com
 5. [Return URLs] (URL を返す) で、ユーザープールのドメインの /oauth2/idpresponse エンドポイントへのパスを入力します。

https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
 6. [Next] (次へ) を選択し、[Done] (完了) を選択します。ドメインを検証する必要はありません。
 7. [Continue] (続行) を選択し、次に [Save] (保存) を選択します。
14. 左のナビゲーションペインで [Keys] (キー) を選択します。
15. [Keys (キー)] ページで、[+] アイコンを選択します。
16. [Register a New Key] ページで、次の操作を行います。
 1. [Key Name] (キー名) に、キー名を入力します。
 2. [Sign In with Apple] (Apple でサインイン) を選択後、[Configure] (設定) を選択します。

3. [Configure Key] (キーの設定) ページで、プライマリアプリ ID として前に作成したアプリ ID を選択します。[保存] を選択します。
4. [Continue] (続行) を選択し、[Register] (登録) を選択します。
17. [Download Your Key] (鍵のダウンロード) ページで、[Download] (ダウンロード) をクリックしてプライベートキーをダウンロードしてから、表示される [Key ID] (キー ID) を書き留め、[Done] (完了) を選択します。このプライベートキーと、このページに表示されている [Key ID] (キー ID) の値は、[ソーシャル IdP を使用してユーザープールを設定する](#) で Apple を ID プロバイダーとして選択した後で必要になります。

ソーシャル IdP を使用してユーザープールを設定する

を使用してユーザープールソーシャル IdP を設定するには AWS マネジメントコンソール

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [ユーザープール] を選択します。
3. リストから既存のユーザープールを選択するか、ユーザープールを新規作成します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択し、[ID プロバイダーを追加] を選択します。
5. ソーシャル IdP ([Facebook]、[Google]、[Login with Amazon]、[Sign in with Apple]) を選択します。
6. ソーシャル IdP の選択に基づいて、次のステップから選択します。
 - Google と Login with Amazon (Amazon でのログイン) — 前のセクションで生成された [app client ID] (アプリケーションクライアント ID) と [app client secret] (アプリケーションクライアントシークレット) を入力します。
 - Facebook — 前のセクションで生成された [app client ID] (アプリケーションクライアント ID) と [app client secret] (アプリケーションクライアントシークレット) を入力し、API バージョン (例えば、バージョン 2.12) を選択します。Facebook API の各バージョンにはライフサイクルとサポート終了日があるため、可能な限り最新のバージョンを選択することをお勧めします。Facebook のスコープと属性は API バージョンによって異なる場合があります。Facebook でソーシャル ID ログインをテストして、フェデレーションが意図したとおり機能することを確認することをお勧めします。

- Sign In with Apple (Apple でのサインイン) — 以前のセクションで作成された [Services ID] (サービス ID)、[Team ID] (チーム ID)、[Key ID] (キー ID) および [private key] プライベートキーを入力します。
7. 使用したい [Authorized scopes] (承認するスコープ) の名前を入力します。スコープは、アプリでアクセスするユーザー属性 (name や email など) を定義します。Facebook の場合は、コンマで区切る必要があります。Google および Login with Amazon の場合は、スペースで区切って指定します。Sign in with Apple の場合は、アクセスするスコープのチェックボックスをオンにします。

ソーシャル ID プロバイダー	スコープ例
Facebook	public_profile, email
Google	profile email openid
Login with Amazon	profile postal_code
Apple でのサインイン	email name

アプリケーションユーザーは、これらの属性をアプリケーションに提供することに同意するよう求められます。ソーシャルプロバイダーのスコープの詳細については、Google、Facebook、Login with Amazon、および Sign in with Apple のドキュメントを参照してください。

以下は、Sign in with Apple の場合にスコープが返らない可能性があるユーザーシナリオです。

- エンドユーザーが Apple のサインインページを離れるとエラーが発生する (Amazon Cognito の内部障害またはデベロッパーが記述したプログラムが原因である可能性があります)
 - サービス ID 識別子がユーザープールや他の認証サービス全体で使用されている
 - エンドユーザーが以前にサインインしてから、デベロッパーによってさらにスコープが追加されている (新しい情報は取得されません)
 - デベロッパーによって削除されたユーザーが Apple ID プロファイルからアプリを削除せずに再度サインインしている
8. IdP からユーザープールに属性をマッピングします。詳細は、「[ユーザープール用 ID プロバイダー属性マッピングの特定](#)」を参照してください。
9. [作成] を選択します。

10. [アプリケーションクライアント] メニューで、リストからアプリケーションクライアントを選択します。新しいソーシャル ID プロバイダーをアプリケーションクライアントに追加するには、[ログインページ] タブに移動し、[マネージドログインページの設定] の [編集] を選択します。
11. [Save changes] (変更の保存) をクリックします。

ソーシャル IdP の設定をテストする

アプリケーションでは、ユーザーのクライアントでブラウザを起動し、ユーザーがソーシャルプロバイダーを使用してサインインできるようにする必要があります。前のセクションのセットアップ手順を完了したら、ソーシャルプロバイダーでのサインインをテストします。次の URL の例では、プレフィックスドメインを持つユーザープールのサインインページをロードします。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://www.example.com
```

このリンクは、[アプリケーションクライアント] メニューに移動してアプリケーションクライアントを選択し、[ログインページ] タブに移動して [ログインページを表示] を選択したときに、Amazon Cognito に表示されるページです。ユーザープールドメインの詳細については、「[ユーザープールのドメインを設定する](#)」を参照してください。クライアント ID とコールバック URL など、アプリケーションクライアントの詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

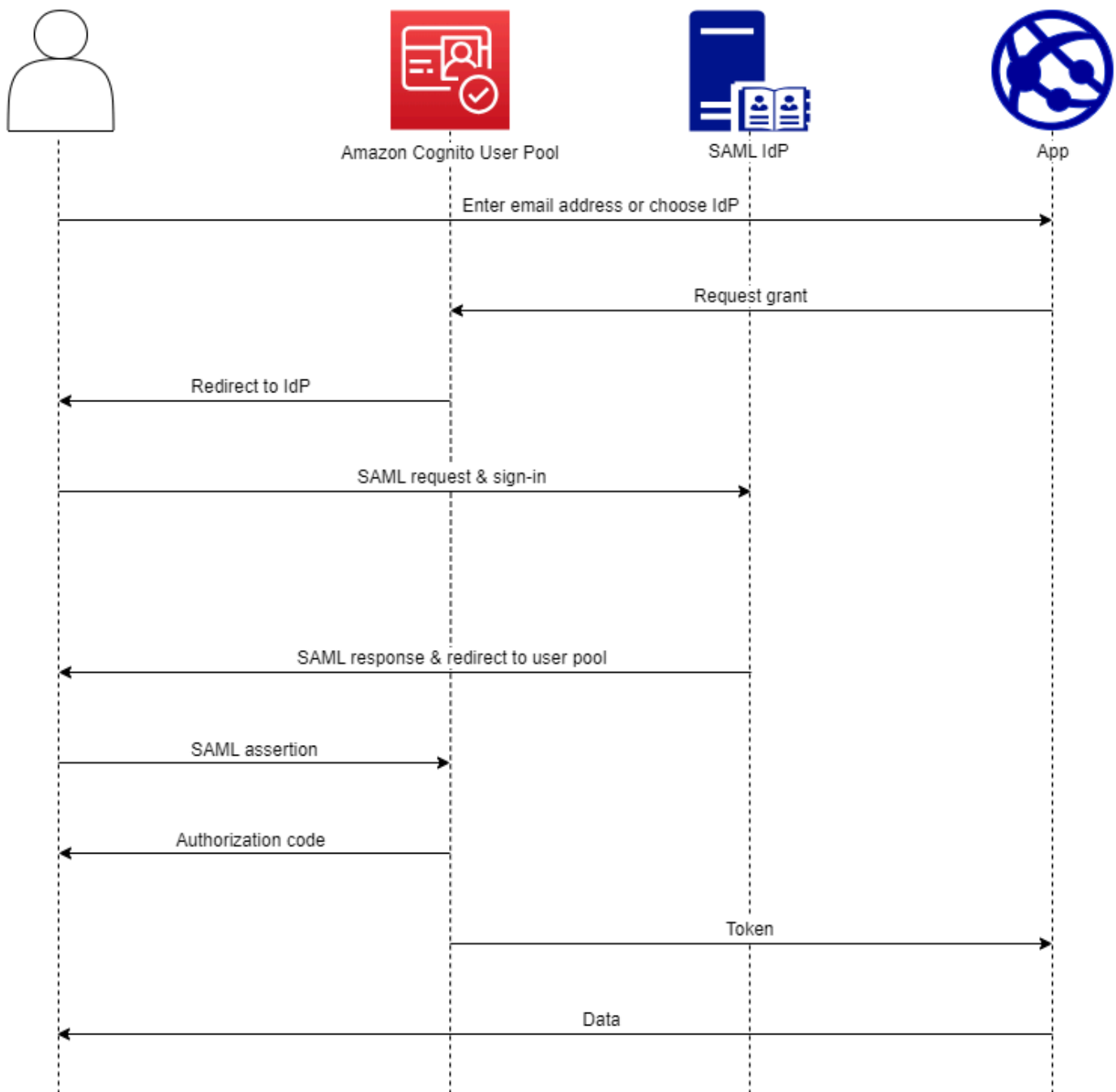
次のリンク例では、identity_provider クエリパラメータを使用して、[認可エンドポイント](#)からソーシャルプロバイダーへのサイレントリダイレクトを設定します。この URL は、マネージドログインによるインタラクティブなユーザープールのサインインをバイパスし、IdP サインインページに直接移動します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/
authorize?identity_provider=Facebook|Google|LoginWithAmazon|
SignInWithApple&response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com
```


ユーザープールによる SAML ID プロバイダーの使用

ウェブアプリケーションおよびモバイルアプリケーションのユーザーに対して、[Microsoft Active Directory フェデレーションサービス \(ADFS\)](#) や [Shibboleth](#) などの SAML ID プロバイダー (IdP) を通じてサインインすることを許可できます。[SAML 2.0 標準](#)をサポートする SAML IdP を選択します。

マネージドログインを使用すると、Amazon Cognito はローカルおよびサードパーティの IdP ユーザーを認証し、JSON ウェブトークン (JWT) を発行します。Amazon Cognito が発行するトークンを使用すると、複数の ID ソースを、すべてのアプリケーションにわたるユニバーサル OpenID Connect (OIDC) 標準に統合できます。Amazon Cognito は、サードパーティープロバイダーからの SAML アサーションを、その SSO 標準に処理できます。SAML IdP は、AWS マネジメントコンソール、AWS CLI または Amazon Cognito ユーザープール API を使用して作成および管理できます。で最初の SAML IdP を作成するには AWS マネジメントコンソール、「」を参照してください[ユーザープールでの SAML ID プロバイダーの追加と管理](#)。

**Note**

サードパーティー IdP を通じたサインインによるフェデレーションは、Amazon Cognito ユーザープールの機能です。Amazon Cognito フェデレーテッド ID と呼ばれる Amazon Cognito アイデンティティプールは、各アイデンティティプールで個別に設定する必要があるフェデレーションの実装です。ユーザープールは、アイデンティティプールへのサード

パーティー IdP にすることができます。詳細については、「[Amazon Cognito アイデンティティプール](#)」を参照してください。

IdP 設定のクイックリファレンス

リクエストを受け入れ、ユーザープールにレスポンスを送信するように SAML IdP を設定する必要があります。ユーザープールを SAML 2.0 IdP の依拠しているパーティーまたはアプリケーションとして追加する方法に関する情報は、SAML IdP のドキュメントに含まれています。次のドキュメントでは、SP エンティティ ID とアサーションコンシューマーサービス (ACS) URL に対して指定する必要がある値を示します。

ユーザープール SAML 値のクイックリファレンス

SP エンティティ ID

```
urn:amazon:cognito:sp:us-east-1_EXAMPLE
```

ACS URL

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

ID プロバイダーをサポートするためにユーザープールを設定する必要があります。外部 SAML IdP を追加する大まかなステップは次のとおりです。

1. IdP から SAML メタデータをダウンロードするか、メタデータエンドポイントへの URL を取得します。「[サードパーティー SAML ID プロバイダーの設定](#)」を参照してください。
2. ユーザープールに新しい IdP を追加します。SAML メタデータをアップロードするか、メタデータ URL を指定します。「[ユーザープールでの SAML ID プロバイダーの追加と管理](#)」を参照してください。
3. IdP をアプリケーションクライアントに割り当てます。「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

トピック

- [Amazon Cognito ユーザープールの SAML IdP について知っておくべきこと](#)
- [SAML ユーザー名の大文字と小文字の区別](#)

- [サードパーティー SAML ID プロバイダーの設定](#)
- [ユーザープールでの SAML ID プロバイダーの追加と管理](#)
- [Amazon Cognito ユーザープールでの SAML セッション開始](#)
- [シングルサインアウトで SAML ユーザーをサインアウトする](#)
- [SAML 署名と暗号化](#)
- [SAML ID プロバイダーの名前と識別子](#)

Amazon Cognito ユーザープールの SAML IdP について知っておくべきこと

SAML 2.0 IdP の実装には、いくつかの要件と制限があります。IdP を実装する場合は、このセクションを参照してください。また、ユーザープールを使用した SAML フェデレーション中に生じるエラーのトラブルシューティングに役立つ情報もあります。

Amazon Cognito は SAML アサーションを処理する

Amazon Cognito ユーザープールは、POST バインドエンドポイントで SAML 2.0 フェデレーションをサポートします。これにより、ユーザープールがユーザーエージェント経由で IdP から直接 SAML レスポンスを受信するため、アプリケーションで SAML アサーションレスポンスを受信して解析する必要がなくなります。ユーザープールはアプリケーションのためのサービスプロバイダー (SP) として機能します。Amazon Cognito は、「[SAML V2.0 Technical Overview](#)」のセクション 5.1.2 と 5.1.4 で説明されている、SP または IdP によるシングルサインオン (SSO) をサポートしています。

有効な IdP 署名証明書を提供する

ユーザープールで SAML IdP を設定するときに、SAML プロバイダーメタデータの署名証明書の有効期限が切れないようにする必要があります。

ユーザープールが複数の署名証明書をサポートする

SAML IdP の SAML メタデータに複数の署名証明書が含まれている場合、サインイン時に、SAML アサーションが SAML メタデータ内のいずれかの証明書と一致すると、ユーザープールは SAML アサーションが有効であると判断します。各署名証明書の長さは 4,096 文字以下にする必要があります。

リレー状態パラメータを維持する

Amazon Cognito と SAML IdP は、セッション情報を relayState パラメータで管理します。

1. Amazon Cognito は、80 バイトを超える relayState 値をサポートします。SAML 仕様では、relayState の値は「長さが 80 バイトを超えてはならない」と規定されていますが、現

在の業界の慣行はこの動作から外れていることがよくあります。その結果、80 バイトを超える relayState 値を拒否すると、多くの標準的な SAML プロバイダーの統合は壊れます。

2. relayState トークンは、Amazon Cognito が管理する状態情報への不透明な参照です。Amazon Cognito は relayState パラメータの内容を保証しません。その内容を解析してアプリケーションでその結果に依存することは避けてください。詳細については、「[SAML 2.0 の仕様](#)」を参照してください。

ACS エンドポイントを特定する

SAML ID プロバイダーは、アサーションコンシューマーエンドポイントを設定することを要求します。IdP は、SAML アサーションを使用して、このエンドポイントにユーザーをリダイレクトします。SAML ID プロバイダーの SAML 2.0 POST バインド用に、ユーザープールドメインで次のエンドポイントを設定します。

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

With an Amazon Cognito domain:

```
https://mydomain.auth.us-east-1.amazoncognito.com/saml2/idpresponse
```

With a custom domain:

```
https://auth.example.com/saml2/idpresponse
```

ユーザープールドメインの詳細については、「[ユーザープールのドメインを設定する](#)」を参照してください。

アサーションの再生ができない

Amazon Cognito の saml2/idpresponse エンドポイントに対して SAML アサーションを繰り返したり、再生したりすることはできません。SAML アサーションを作成すると、そのアサーション ID は以前の IdP レスポンスの ID と重複します。

ユーザープール ID は SP エンティティ ID である

IdP については、オーディエンス URI または SP エンティティ ID ととも呼ばれるサービスプロバイダー (SP) urn のユーザープール ID で指定する必要があります。ユーザープールのオーディエンス URI の形式は次のとおりです。

```
urn:amazon:cognito:sp:us-east-1_EXAMPLE
```

ユーザープール ID は、[Amazon Cognito コンソール](#)の [ユーザープールの概要] で確認できます。

必要なすべての属性をマッピングする

ユーザープールで必須として設定した属性の値を入力するように SAML IdP を設定します。例えば、email はユーザープールの一般的な必須属性です。ユーザーがサインインする前に、ユーザープール属性 email にマッピングするクレームを SAML IdP アサーションに含める必要があります。属性のマッピングの詳細については、「[IdP 属性をプロフィールとトークンにマッピングする](#)」を参照してください。

アサーション形式には特定の要件がある

SAML IdP には、SAML アサーションに次のクレームを含める必要があります。

- NameID クレーム。Amazon Cognito は、NameID によって SAML アサーションを送信先ユーザーと関連付けます。NameID が変更になった場合、Amazon Cognito はアサーションを新しいユーザー用と見なします。IdP 設定で NameID に設定する属性には永続的な値が必要です。SAML ユーザーをユーザープール内の一貫したユーザープロフィールに割り当てするには、変更されない値を持つ属性からの NameID クレームを割り当てます。

```
<saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:persistent">
  carlos
</saml2:NameID>
```

IdP NameID クレームの Format で urn:oasis:names:tc:SAML:1.1:nameid-format:persistent は、IdP が変更のない値を渡していることを示します。Amazon Cognito は、この形式の宣言を必要とせず、IdP が NameID クレームの形式を指定しない場合、urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified の形式を割り当てます。この動作は、[SAML 2.0 仕様](#)のセクション 2.2.2 (複合タイプ NameIDType) に準拠しています。

- ユーザープールの SP エンティティ ID をレスポンスのターゲットとして設定する値 Audience を含む AudienceRestriction クレーム。

```
<saml:AudienceRestriction>
  <saml:Audience> urn:amazon:cognito:sp:us-east-1_EXAMPLE
</saml:AudienceRestriction>
```

- SP が開始したシングルサインオンの場合、元の SAML リクエスト ID の InResponseTo 値を持つ Response 要素。

```
<saml2p:Response Destination="https://mydomain.auth.us-east-1.amazoncognito.com/
saml2/idpresponse" 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 が開始した SAML アサーションに InResponseTo 値を含めることはできません。

- ユーザープール saml2/idpresponse エンドポイントの Recipient 値と、SP が開始した SAML の場合の、元の SAML リクエスト ID と一致する InResponseTo 値を含む SubjectConfirmationData 要素。

```
<saml2:SubjectConfirmationData InResponseTo="_dd0a3436-bc64-4679-a0c2-
cb4454f04184" NotOnOrAfter="Date-time stamp" Recipient="https://mydomain.auth.us-
east-1.amazoncognito.com/saml2/idpresponse"/>
```

SP が開始したサインインリクエスト

[認可エンドポイント](#)がユーザーを IdP サインインページにリダイレクトすると、Amazon Cognito は HTTP GET リクエストの URL パラメータに SAML リクエストを含めます。SAML リクエストには、ACS エンドポイントなどの、ユーザープールに関する情報が含まれます。オプションで、これらのリクエストに暗号化署名を適用できます。

リクエストへの署名とレスポンスの暗号化

SAML プロバイダーを持つすべてのユーザープールは、Amazon Cognito が SAML リクエストに割り当てるデジタル署名の非対称キーペアと署名証明書を生成します。暗号化された SAML レスポンスをサポートするように設定したすべての外部 SAML IdP により、Amazon Cognito は、そのプロバイダーの新しいキーペアと暗号化証明書を生成します。パブリックキーを使用して証明書の表示とダウンロードを行うには、Amazon Cognito コンソールの [ソーシャルプロバイダーと外部プロバイダー] メニューで IdP を選択します。

ユーザープールからの SAML リクエストとの信頼を確立するには、ユーザープールの SAML 2.0 署名証明書のコピーを IdP に提供します。署名付きリクエストを受け入れるように IdP を設定しない場合、IdP はユーザープールが署名した SAML リクエストを無視する可能性があります。

1. Amazon Cognito は、ユーザーが IdP に渡す SAML リクエストにデジタル署名を適用します。ユーザープールは、すべてのシングルログアウト (SLO) リクエストに署名しますが、SAML 外部 IdP のシングルサインオン (SSO) リクエストに署名するようにユーザープールを設定できます。証明書のコピーを提供すると、IdP はユーザーの SAML リクエストの整合性を検証できます。

2. SAML IdP は、暗号化証明書を使用して SAML レスポンスを暗号化できます。SAML 暗号化を使用して IdP を設定する場合、IdP は、暗号化されたレスポンスのみを送信する必要があります。

アルファベット以外の文字のエンコードを行う

Amazon Cognito では、IdP が属性値として渡す 4 バイトの UTF-8 文字 (# や # など) は受け付けません。文字を Base 64 にエンコードしてテキストとして渡し、アプリでデコードできます。

次の例では、属性のクレームは受け付けられません。

```
<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>
```

上記の例とは対照的に、次の属性のクレームは受け付けられます。

```
<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>
```

メタデータエンドポイントには、有効な Transport Layer Security (TLS) が必要である

HTTPS メタデータエンドポイント URL を使用して SAML IdP を作成中に `InvalidParameterException` (例えば「Error retrieving metadata from *<metadata endpoint>*」) が表示される場合、メタデータエンドポイントの SSL が正しくセットアップされていること、および有効な SSL 証明書が関連付けられていることを確認してください。詳細については、「[SSL/TLS 証明書とは何ですか?](#)」を参照してください。

メタデータエンドポイントは HTTP または HTTPS の標準 TCP ポート上に存在する必要がある

Amazon Cognito は、HTTP の場合は標準 TCP ポート 80、HTTPS の場合は 443 のみ、SAML プロバイダーのメタデータ URL を受け入れます。セキュリティのベストプラクティスとして、SAML メタデータは、https:// プレフィックスを付けた TLS 暗号化 URL でホストします。メタデータ URL は、*http://www.example.com/saml2/metadata.xml* または *https://www.example.com/saml2/metadata.xml* の形式で入力します。Amazon Cognito コンソールは、https:// プレフィックスが付いたメタデータ URL のみを受け入れます。

す。[CreateIdentityProvider](#) と [UpdateIdentityProvider](#) を使用して IdP メタデータを設定することもできます。

IdP が開始した SAML を持つアプリケーションクライアントは SAML でのみサインインできる

アプリケーションクライアントで、IdP が開始した署名をサポートする SAML 2.0 IdP のサポートを有効にすると、そのアプリケーションクライアントに追加できるのは他の SAML 2.0 IdP のみになります。この方法で設定されたアプリケーションクライアントには、ユーザープール内のユーザーディレクトリとすべての非 SAML 外部 ID プロバイダーを追加することはできません。

ログアウトレスポンスには POST バインディングを使用する必要がある

/saml2/logout エンドポイントは、HTTP POST リクエストとして LogoutResponse を受け入れます。ユーザープールは、HTTP GET バインディングによるログアウトレスポンスを受け入れません。

メタデータ署名証明書のローテーション

URL でメタデータを提供すると、Amazon Cognito は、SAML メタデータを最大 6 時間キャッシュします。メタデータ署名証明書のローテーションを実行する場合は、元の証明書と新しい証明書の両方を少なくとも 6 時間公開するようにメタデータソースを設定します。Amazon Cognito がメタデータ URL からキャッシュを更新すると、各証明書は有効と見なされ、SAML IdP は新しい証明書を使用して SAML アサーションの署名を開始できます。この期間が経過したら、公開したメタデータから元の証明書を削除できます。

SAML ユーザー名の大文字と小文字の区別

フェデレーションユーザーがサインインしようとする、SAML ID プロバイダー (IdP) はユーザーの SAML アサーションで Amazon Cognito に一意の NameId を渡します。Amazon Cognito は、SAML フェデレティッドユーザーを NameId クレームによって識別します。ユーザープールの大文字と小文字の区別の設定にかかわらず、Amazon Cognito では、大文字と小文字を区別する一意の NameId クレームを渡すときに、SAML IdP の既存フェデレーションユーザーを認識します。email のような属性を NameId にマッピングし、ユーザーが E メールアドレスを変更すると、アプリケーションにサインインできません。

変更されない値を持つ IdP 属性から SAML アサーションに NameId をマッピングします。

例えば、Carlos は、Carlos@example.com の NameId 値を渡した Active Directory フェデレーションサービス (ADFS) SAML アサーションからの大文字と小文字を区別しないユーザープールにユーザープロファイルを持っているとします。次回 Carlos がサインインしようとする、ADFS IdP は

carlos@example.com の NameId 値を渡します。NameId は大文字と小文字が完全に一致する必要があるため、サインインは成功しません。

NameId を変更した後、ユーザーがログインできない場合は、ユーザープールからそのユーザーのプロファイルを削除してください。Amazon Cognito は、次回サインインしたときに新しいユーザープロフィールを作成します。

トピック

- [サードパーティー SAML ID プロバイダーの設定](#)
- [ユーザープールでの SAML ID プロバイダーの追加と管理](#)
- [Amazon Cognito ユーザープールでの SAML セッション開始](#)
- [シングルサインアウトで SAML ユーザーをサインアウトする](#)
- [SAML 署名と暗号化](#)
- [SAML ID プロバイダーの名前と識別子](#)

サードパーティー SAML ID プロバイダーの設定

SAML ID プロバイダー (IdP) をユーザープールに追加する場合は、IdP の管理インターフェイスで設定を更新する必要があります。このセクションでは、IdP に提供する必要がある値をフォーマットする方法について説明します。また、IdP とそのユーザープールへの SAML クレームを識別する静的なまたはアクティブな URL メタデータドキュメントを取得する方法についても説明します。

Amazon Cognito ユーザープールのフェデレーションと連携するようにサードパーティー SAML 2.0 ID プロバイダー (IdP) ソリューションを設定するには、次のアサーションコンシューマーサービス (ACS) の URL にリダイレクトするように SAML IdP を設定する必要があります:

<https://mydomain.auth.us-east-1.amazoncognito.com/saml2/idpresponse>。ユーザープールに Amazon Cognito ドメインがある場合は、[Amazon Cognito コンソール](#)のユーザープールの [ドメイン] メニューでユーザープールドメインのパスを見つけることができます。

一部の SAML IdP では、オーディエンス URI または SP エンティティ ID と呼ばれる urn を `urn:amazon:cognito:sp:us-east-1_EXAMPLE` の形式で指定する必要があります。ユーザープール ID は、Amazon Cognito コンソールの [ユーザープールの概要] で確認できます。

また、SAML IdP を設定して、ユーザープールの必須属性として指定した属性の値を指定する必要があります。通常、email はユーザープールに必要な属性です。この場合、SAML IdP は、SAML アサーションに何らかの形式の email クレームを指定し、そのクレームをそのプロバイダーの属性にマッピングする必要があります。

サードパーティーの SAML 2.0 IdP ソリューションについての以下の設定情報は、Amazon Cognito ユーザープールでフェデレーションのセットアップを開始するのに適しています。最新情報については、プロバイダーのドキュメントを直接参照してください。

SAML リクエストに署名するには、ユーザープール署名証明書によって署名されたリクエストを信頼するように IdP を設定する必要があります。暗号化された SAML レスポンスを受け入れるには、ユーザープールへのすべての SAML レスポンスを暗号化するように IdP を設定する必要があります。プロバイダーは、これらの機能の設定に関するドキュメントを用意します。Microsoft の例については、「[Microsoft Entra の SAML トークン暗号化を構成する](#)」を参照してください。

 Note

Amazon Cognito では、ID プロバイダーのメタデータドキュメントのみが必要です。プロバイダーによっては、IAM または AWS IAM アイデンティティセンターとの SAML 2.0 フェデレーション用にカスタマイズした設定情報を提供している場合もあります。Amazon Cognito 統合を設定する方法については、メタデータに関するドキュメントを取得する一般的な手順を確認し、ユーザープールで残りの設定を管理してください。

ソリューション	詳細情報
Microsoft Entra ID	フェデレーションメタデータ
Okta	SAML アプリケーション統合の IdP メタデータと SAML 署名証明書をダウンロードする方法
Auth0	Auth0 を SAML ID プロバイダーとして設定する
Ping Identity (PingFederate)	PingFederate からの SAML メタデータのエクスポート
JumpCloud	SAML 設定に関する注意事項
SecureAuth	SAML アプリケーション統合

ユーザープールでの SAML ID プロバイダーの追加と管理

Amazon Cognito を操作するように ID プロバイダーを設定したら、ユーザープールとアプリケーションクライアントに追加できます。次の手順は、Amazon Cognito ユーザープールで SAML プロバイダーの作成、変更、削除を行う方法を示しています。

AWS マネジメントコンソール

を使用して、SAML ID プロバイダー (IdPs AWS マネジメントコンソール を作成および削除できます。

SAML IdP を作成する前に、サードパーティー IdP から取得した SAML メタデータドキュメントが必要です。必要な SAML メタデータドキュメントの取得または生成方法については、「[サードパーティー SAML ID プロバイダーの設定](#)」を参照してください。

ユーザープールに SAML 2.0 IdP を設定するには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択し、[ID プロバイダーを追加] を選択します。
5. [SAML] IdP を選択します。
6. プロバイダー名を入力します。このわかりやすい名前を、identity_provider リクエストパラメータで [認可エンドポイント](#) に渡すことができます。
7. カンマで区切られた [Identifiers] (識別子) を入力します。識別子は Amazon Cognito に、ユーザーがサインインしたときに入力したメールアドレスを確認し、ドメインに対応するプロバイダーに誘導する必要があることを伝えます。
8. ユーザーがログアウトしたときに、Amazon Cognito が署名されたサインアウト要求をプロバイダーに送信するためには、[Add sign-out flow] (サインアウトフローの追加) を選択します。マネージドログインの設定時に作成した <https://mydomain.auth.us-east-1.amazoncognito.com/saml2/logout> エンドポイントにサインアウトレスポンスを送信するよう、SAML 2.0 IdP を設定する必要があります。saml2/logout エンドポイントでは、ポストバインディングを使用します。

Note

このオプションを選択し、SAML IdP が署名付きログアウトリクエストを期待する場合は、SAML IdP に対して、ユーザープールからの署名証明書を提供する必要もあります。

SAML IdP は署名されたログアウトリクエストを処理し、Amazon Cognito セッションからユーザーをサインアウトさせます。

9. [IdP が開始した SAML サインイン] 設定を選択します。セキュリティのベストプラクティスとして、[SP が開始した SAML アサーションのみを受け入れる] を選択します。未承諾の SAML サインインセッションを安全に受け入れる環境を準備している場合は、[SP が開始した/IdP が開始した SAML アサーションを受け入れる] を選択します。詳細については、「[Amazon Cognito ユーザープールでの SAML セッション開始](#)」を参照してください。
10. [Metadata document source] (メタデータドキュメントソース) を選択します。IdP がパブリック URL で SAML メタデータを提供する場合は、[Metadata document URL] (メタデータドキュメント URL) を選択してそのパブリック URL を入力できます。それ以外の場合は、[Upload metadata document] (メタデータドキュメントをアップロード) を選択し、プロバイダーから以前ダウンロードしたメタデータファイルを選択します。

Note

プロバイダーにパブリックエンドポイントがある場合は、ファイルをアップロードするのではなく、メタデータドキュメントの URL を入力することをお勧めします。Amazon Cognito は、メタデータ URL からのメタデータを自動的に更新します。通常、メタデータの更新は 6 時間ごとまたはメタデータの有効期限が切れる前のいずれか早いタイミングで発生します。

11. [SAML プロバイダーとユーザープール間で属性をマッピング] をクリックして、SAML プロバイダーの属性をユーザープールのユーザープロファイルにマッピングします。ユーザープールの必須属性を属性マップに含めます。

例えば、[User pool attribute] (ユーザープール属性) email を選択する場合、IdP からの SAML アサーションに表示される SAML 属性名を入力します。IdP がサンプル SAML アサーションを提供している場合は、これらのサンプルアサーションを使用して名前を見つけやすいかもしれません。email などの単純な名前を使用している IdP もあれば、次のような名前を使用している IdP もあります。

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

12. [作成] を選択します。

API/CLI

SAML ID プロバイダー (IdP) を作成して管理するには、以下のコマンドを使用します。

IdP を作成し、メタデータドキュメントをアップロードする

- AWS CLI: `aws cognito-idp create-identity-provider`

メタデータファイルの例: `aws cognito-idp 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`

ここでは `details.json` に以下が含まれます。

```
"ProviderDetails": {
  "MetadataFile": "<SAML metadata XML>",
  "IDPSignout" : "true",
  "RequestSigningAlgorithm" : "rsa-sha256",
  "EncryptedResponses" : "true",
  "IDPInit" : "true"
}
```

Note

<SAML ##### XML> に文字 " のインスタンスが含まれている場合は、エスケープ文字の \ を追加して、次のようにする必要があります: \"。

メタデータ URL の例: `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/sso/`

```
saml/metadata --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
```

- AWS API: [CreateIdentityProvider](#)

IdP の新規メタデータドキュメントをアップロードする

- AWS CLI: `aws cognito-idp update-identity-provider`

メタデータファイルの例: `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`

ここでは details.json に以下が含まれます。

```
"ProviderDetails": {
  "MetadataFile": "<SAML metadata XML>",
  "IDPSignout" : "true",
  "RequestSigningAlgorithm" : "rsa-sha256",
  "EncryptedResponses" : "true",
  "IDPInit" : "true"
}
```

 Note

<SAML ##### XML> に文字 " のインスタンスが含まれている場合は、エスケープ文字の \ を追加して、次のようにする必要があります: \"。

メタデータ URL の例: `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](#)

固有の IdP に関する情報を取得するには

- 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: [DescribeIdentityProvider](#)

すべての IdP に関する情報を一覧表示するには

- AWS CLI: `aws cognito-idp list-identity-providers`

```
例: aws cognito-idp list-identity-providers --user-pool-id us-east-1_EXAMPLE --max-results 3
```

- AWS API: [ListIdentityProviders](#)

IdP を削除する

- 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](#)

ユーザープールを証明書利用者として追加するために SAML IdP をセットアップする

- ユーザープールのサービスプロバイダー URN は `urn:amazon:cognito:sp:us-east-1_EXAMPLE` です。Amazon Cognito には、SAML レスポンスでこの URN に一致するオーディエンス制限値が必要です。お客様の IdP は、次の POST バインディングエンドポイントを IdP-to-SP 応答メッセージに使用します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/saml2/idpresponse
```

- SAML IdP によって、SAML アサーションの NameID およびユーザープール必須属性が入力されている必要があります。NameID は、ユーザープールの SAML フェデレーションユーザーを一意に識別するために使用されます。IdP は、各ユーザーの SAML 名 ID を、大文字と小文字を区別する一貫した形式で渡す必要があります。ユーザー名 ID の値にバリエーションがあると、新しいユーザープロファイルが作成されます。

SAML 2.0 IDP にデジタル署名用証明書を提供するには

- IdP が SAML ログアウトリクエストの検証に使用できるパブリックキーのコピーを Amazon Cognito からダウンロードするには、ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューを選択して IdP を選択し、[署名用証明書を表示] で [.crt としてダウンロード] を選択します。

Amazon Cognito コンソールを使用して、ユーザープールで設定した SAML プロバイダーを削除できます。

SAML プロバイダーを削除する

1. [Amazon Cognito コンソール](#)にサインインします。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択します。
4. 削除する SAML IdP の横のラジオボタンを選択します。
5. [Delete identity provider] (ID プロバイダーの削除) のプロンプトが表示されたら、SAML プロバイダー名を入力して削除を確認し、[Delete] (削除) を選択します。

Amazon Cognito ユーザープールでの SAML セッション開始

Amazon Cognito は、サービスプロバイダーが開始した (SP が開始した) シングルサインオン (SSO) と IdP が開始した SSO をサポートしています。セキュリティのベストプラクティスとして、SP が開始した SSO をユーザープールに実装します。[SAML V2.0 技術概要](#)のセクション 5.1.2 では、SP で開始される SSO について説明します。Amazon Cognito は、アプリケーションの ID プロバイダー (IdP) です。アプリケーションは、認証されたユーザーのトークンを取得するサービスプロバイダー (SP) です。ただし、サードパーティー IdP を使用してユーザーを認証する場合、Amazon Cognito は SP です。SAML 2.0 ユーザーが、SP が開始したフローで認証する場合、常に Amazon Cognito にリクエストを行い、認証のために IdP にリダイレクトする必要があります。

一部のエンタープライズユースケースでは、内部アプリケーションへのアクセスは Enterprise IdP がホストするダッシュボードのブックマークから始まります。ユーザーがブックマークを選択すると、IdP は SAML 応答を生成して SP に送信し、アプリケーションでユーザーを認証します。

IdP が開始した SSO をサポートするように、ユーザープールで SAML IdP を設定できます。IdP が開始した認証をサポートする場合、Amazon Cognito は SAML リクエストで認証を開始しないため、

受信した SAML レスポンスを要請したことを検証できません。SP が開始した SSO では、Amazon Cognito は、元のリクエストに対して SAML レスポンスを検証する状態パラメータを設定します。SP が開始したサインインを使用すると、クロスサイトリクエストフォージェリ (CSRF) から保護することもできます。

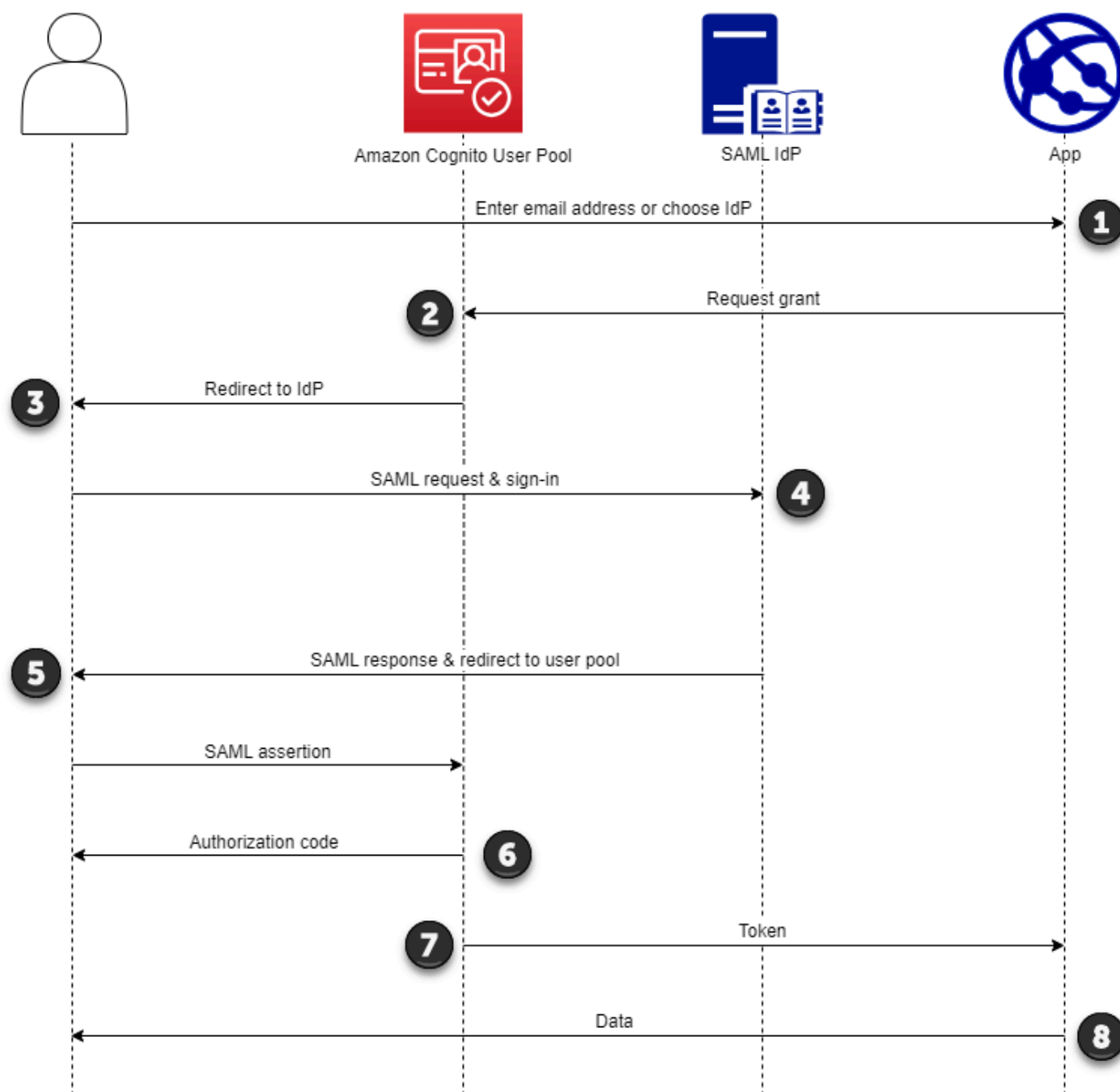
トピック

- [SP が開始した SAML サインインを実装する](#)
- [IdP が開始した SAML サインインを実装する](#)

SP が開始した SAML サインインを実装する

ベストプラクティスとして、ユーザープールに、サービスプロバイダーが開始した (SP が開始した) サインインを実装します。Amazon Cognito はユーザーのセッションを開始し、IdP にリダイレクトします。この方法を使用すると、サインインリクエストを提示するユーザーの制御が最大になります。特定の条件下で、IdP が開始したサインインを許可することもできます。

以下のプロセスは、ユーザーが SAML プロバイダーを介してユーザープールへの SP が開始したサインインを完了する方法を示しています。



1. ユーザーはサインインページで E メールアドレスを入力します。ユーザーを IdP にリダイレクトすることを決めるには、カスタム構築したアプリケーションで E メールアドレスを収集するか、ウェブビューでマネージドログインを呼び出すことができます。

IdP を一覧表示するか、E メールアドレスの入力を求めて SAML IdP の識別子と照合するように、マネージドログインページを設定できます。E メールアドレスの入力を求めるには、マネー

- ジドログインのブランディングスタイルを編集し、[基盤] で [認証動作] を見つけ、[プロバイダーの表示] で [表示スタイル] を [ドメイン検索入力] に設定します。
- アプリケーションは、ユーザープールリダイレクトエンドポイントを呼び出し、アプリケーションに対応するクライアント ID と、ユーザーに対応する IdP ID を持つセッションをリクエストします。
 - Amazon Cognito は、AuthnRequest 要素内の、[オプションで署名された](#)、SAML リクエストを使用して、ユーザーを IdP にリダイレクトします。
 - IdP は、インタラクティブに、またはブラウザ Cookie で記憶されたセッションを使用して、ユーザーを認証します。
 - IdP は、POST ペイロードで、[オプションで暗号化された](#) SAML アサーションを使用して、ユーザーをユーザープールの SAML レスポンスエンドポイントにリダイレクトします。

 Note

Amazon Cognito は、5 分以内にレスポンスを受信しないセッションをキャンセルし、ユーザーをマネージドログインにリダイレクトします。ユーザーにこの結果が生じると、Something went wrong エラーメッセージを受け取ります。

- SAML アサーションが検証され、レスポンスのクレームから[ユーザー属性がマッピング](#)されると、Amazon Cognito がユーザープールのユーザープロフィールを内部で作成または更新します。通常、ユーザープールはユーザーのブラウザセッションに認可コードを返します。
- ユーザーは、認可コードをアプリケーションに提示します。アプリケーションはコードを JSON ウェブトークン (JWT) と交換します。
- アプリケーションは、ユーザーの ID トークンを認証として受け入れて処理し、アクセストークンを使用してリソースへの認可されたリクエストを生成し、更新トークンを保存します。

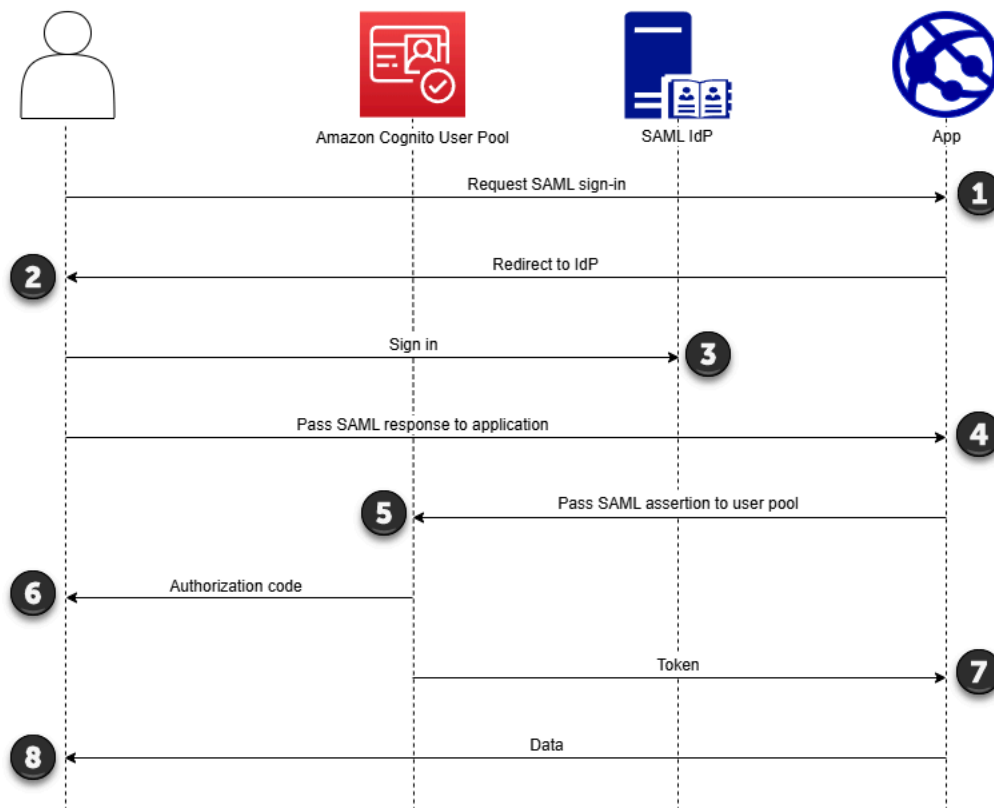
ユーザーが認証して、認可コード付与を受け取ると、ユーザープールは ID、アクセス、更新トークンを返します。ID トークンは、OIDC ベースの ID 管理用の認証オブジェクトです。アクセストークンは、[OAuth 2.0](#) スコープを持つ認可オブジェクトです。更新トークンは、ユーザーの現在のトークンの有効期限が切れたときに新しい ID トークンとアクセストークンを生成するオブジェクトです。ユーザープールアプリケーションクライアントでユーザーのトークンの期間を設定できます。

更新トークンの期間を選択することもできます。ユーザーの更新トークンの有効期限が切れたら、再度サインインする必要があります。SAML IdP を介して認証された場合、ユーザーのセッション期間は、IdP とのセッションの有効期限ではなく、トークンの有効期限によって設定されます。アプリケーションは、各ユーザーの更新トークンを保存し、有効期限が切れたらセッションを更新する必要

があります。マネージドログインは、1 時間有効なブラウザ Cookie でユーザーセッションを維持します。

IdP が開始した SAML サインインを実装する

IdP が開始した SAML 2.0 サインイン用に ID プロバイダーを設定すると、[認可エンドポイント](#) でセッションを開始することなく、ユーザープールドメインの `saml2/idpresponse` エンドポイントに SAML アサーションを提示できます。この設定を持つユーザープールは、リクエストされたアプリケーションクライアントがサポートするユーザープールの外部 ID プロバイダーから、IdP が開始した SAML アサーションを受け入れます。



1. ユーザーがアプリケーションで SAML サインインをリクエストします。
2. アプリケーションは、ブラウザを起動するか、SAML プロバイダーのサインインページにユーザーをリダイレクトします。
3. IdP は、インタラクティブに、またはブラウザ Cookie で記憶されたセッションを使用して、ユーザーを認証します。
4. IdP は、POST 本文の SAML アサーションまたはレスポンスを使用して、ユーザーをアプリケーションにリダイレクトします。

5. アプリケーションは、ユーザープールの `saml2/idpresponse` エンドポイントへのリクエストの POST 本文に SAML アサーションを追加します。
6. Amazon Cognito は、ユーザーに認証コードを発行します。
7. ユーザーは、認可コードをアプリケーションに提示します。アプリケーションはコードを JSON ウェブトークン (JWT) と交換します。
8. アプリケーションは、ユーザーの ID トークンを認証として受け入れて処理し、アクセストークンを使用してリソースへの認可されたリクエストを生成し、更新トークンを保存します。

次の手順では、IdP が開始した SAML 2.0 プロバイダーを使用した設定とサインインの全体的なプロセスについて説明します。

1. ユーザープールとアプリケーションクライアントを作成または指定します。
2. ユーザープールで SAML 2.0 IdP を作成します。
3. IdP の開始をサポートするように IdP を設定します。IdP が開始した SAML では、他の SSO プロバイダーには適用されないセキュリティ上の考慮事項が導入されています。このため、IdP が開始したサインインで SAML プロバイダーを使用するアプリケーションクライアントに、ユーザープール自体を含む非 SAML IdP を追加することはできません。
4. IdP が開始した SAML プロバイダーを、ユーザープール内のアプリケーションクライアントに関連付けます。
5. SAML IdP のサインインページにユーザーを誘導し、SAML アサーションを取得します。
6. SAML アサーションを使用して、ユーザーをユーザープール `saml2/idpresponse` エンドポイントに誘導します。
7. JSON ウェブトークン (JWT) を受け取ります。

ユーザープールで未承諾の SAML アサーションを受け入れるには、アプリケーションセキュリティへの影響を考慮する必要があります。リクエストスプーフィングと CSRF の試行は、IdP が開始したリクエストを受け入れるときに行われる可能性があります。ユーザープールは IdP が開始したサインインセッションを検証できませんが、Amazon Cognito はリクエストパラメータと SAML アサーションを検証します。

さらに、SAML アサーションに `InResponseTo` クレームが含まれておらず、過去 6 分以内に発行されている必要があります。

IdP が開始した SAML を使用してリクエストを `/saml2/idpresponse` に送信する必要があります。SP が開始した認可リクエストおよびマネージドログインの認可リクエストの場合、リクエスト

されたアプリケーションクライアント、スコープ、リダイレクト URI、その他の詳細を識別するパラメータを、HTTP GET リクエストのクエリ文字列パラメータとして指定する必要があります。ただし、IdP が開始した SAML アサーションの場合、リクエストの詳細は、HTTP POST リクエスト本文の RelayState パラメータとしてフォーマットする必要があります。また、リクエスト本文には、SAML アサーションを SAMLResponse パラメータとして含める必要もあります。

次に示すのは、IdP が開始した SAML プロバイダーのリクエストとレスポンスの例です。

```
POST /saml2/idpresponse 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 マネジメントコンソール

IdP が開始した SAML の IdP を設定するには

1. [ユーザープール](#)、[アプリケーションクライアント](#)、および SAML ID プロバイダーを作成します。
2. 関連付けられている場合、すべてのソーシャル ID プロバイダーと OIDC ID プロバイダーについてアプリケーションクライアントから関連付けを解除します。
3. ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューに移動します。
4. SAML プロバイダーを編集または追加します。
5. [IdP が開始した SAML サインイン] で、[SP が開始した/IdP が開始した SAML アサーションを受け入れる] を選択します。
6. [Save changes] (変更の保存) をクリックします。

API/CLI

IdP が開始した SAML の IdP を設定するには

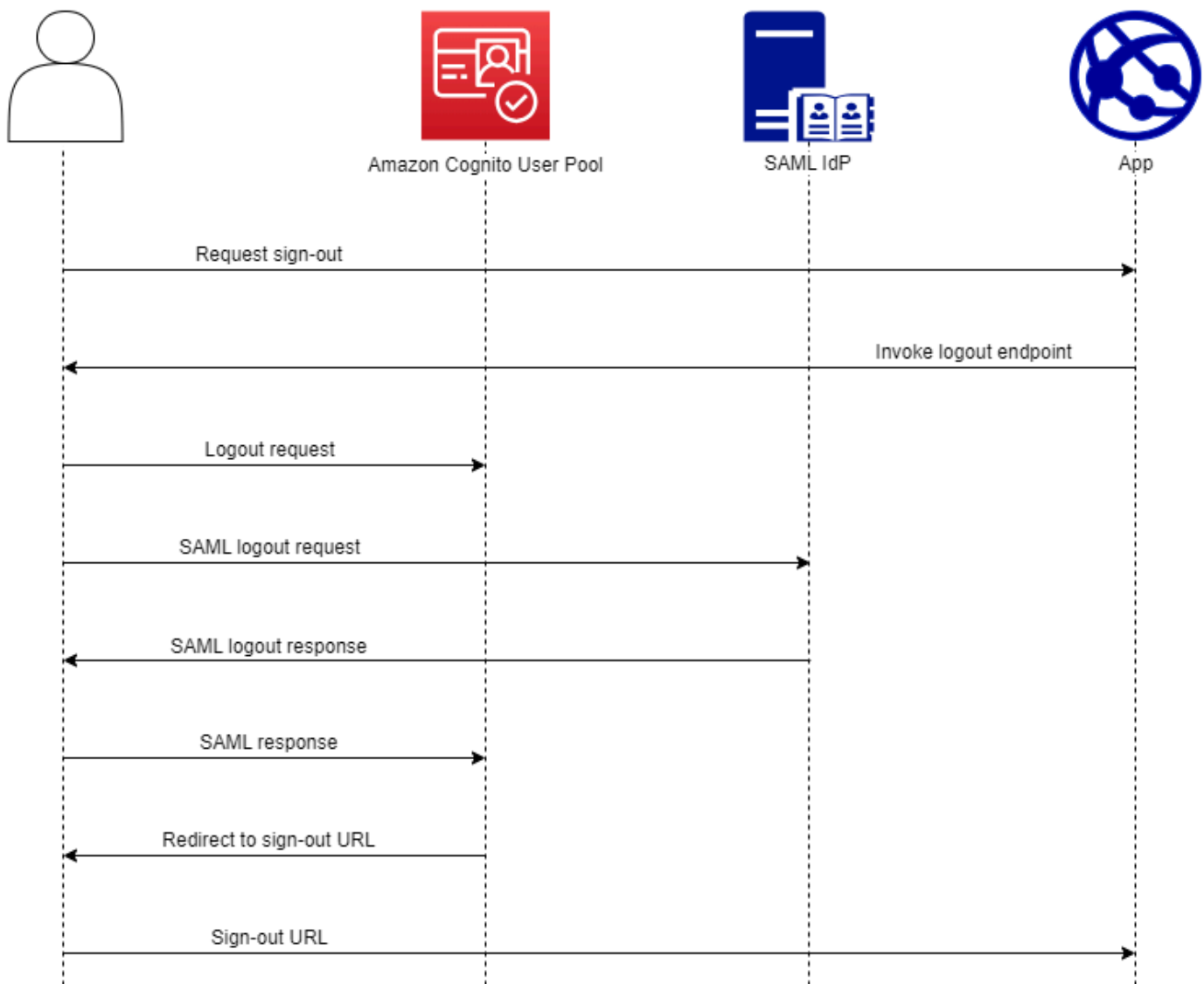
[CreateIdentityProvider](#) API リクエストまたは [UpdateIdentityProvider](#) API リクエストの `IDPInit` パラメータを使用して、IdP が開始した SAML を設定します。以下は、IdP が開始した SAML をサポートする IdP の `ProviderDetails` の例です。

```
"ProviderDetails": {
  "MetadataURL" : "https://myidp.example.com/saml/metadata",
  "IDPSignout" : "true",
  "RequestSigningAlgorithm" : "rsa-sha256",
  "EncryptedResponses" : "true",
  "IDPInit" : "true"
}
```

シングルサインアウトで SAML ユーザーをサインアウトする

Amazon Cognito は SAML 2.0 [シングルログアウト](#) (SLO) をサポートしています。SLO を使用すると、アプリケーションはユーザープールからサインアウトするときに、SAML ID プロバイダー (IdP) からユーザーをサインアウトできます。これにより、ユーザーがアプリケーションに再度サインインする場合は、SAML IdP で認証する必要があります。そうしない場合は、認証情報を提供する要件なくアプリケーションに渡す、IdP またはユーザープールブラウザの Cookie が設置される可能性があります。

サインアウトフローをサポートするように SAML IdP を設定すると、Amazon Cognito は、署名付き SAML ログアウトリクエストを使用してユーザーを IdP にリダイレクトします。Amazon Cognito は、IdP メタデータの `SingleLogoutService` URL からリダイレクト場所を決定します。Amazon Cognito は、ユーザープール署名証明書を使用してサインアウトリクエストに署名します。



SAML セッションを持つユーザーをユーザープール /logout エンドポイントに誘導すると、Amazon Cognito は、次のリクエストで、IdP メタデータで指定された SLO エンドポイントに SAML ユーザーをリダイレクトします。

```

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]
  
```

その後、ユーザーは IdP から LogoutResponse を使用して saml2/logout エンドポイントに戻ります。IdP は HTTP POST リクエストで LogoutResponse を送信する必要があります。その後、Amazon Cognito は、最初のサインアウトリクエストからリダイレクト先にリダイレクトします。

SAML プロバイダーは、複数の AuthnStatement を含む LogoutResponse を送信する場合があります。このタイプのレスポンスの最初の AuthnStatement の sessionIndex は、最初にユーザーを認証した SAML レスポンスの sessionIndex と一致する必要があります。sessionIndex が他の AuthnStatement にある場合、Amazon Cognito はセッションを認識せず、ユーザーはサインアウトされません。

AWS マネジメントコンソール

SAML サインアウトを設定するには

1. [ユーザープール](#)、[アプリケーションクライアント](#)、および SAML IdP を作成します。
2. SAML ID プロバイダーを作成または編集するときは、[ID プロバイダーに関する情報] で、[サインアウトフローを追加] というタイトルのチェックボックスをオンにします。
3. ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューで IdP を選択し、[署名証明書] を見つけます。
4. [.crt としてダウンロード] を選択します。
5. SAML シングルログアウトをサポートし、署名をサポートするように SAML プロバイダーを設定し、ユーザープール署名証明書をアップロードします。IdP は、ユーザープールドメインの /saml2/logout にリダイレクトする必要があります。

API/CLI

SAML サインアウトを設定するには

[CreateIdentityProvider](#) API リクエストまたは [UpdateIdentityProvider](#) API リクエストの IDPSignout パラメータを使用してシングルログアウトを設定します。以下は、SAML シングルログアウトをサポートする IdP の ProviderDetails の例です。

```
"ProviderDetails": {
  "MetadataURL" : "https://myidp.example.com/saml/metadata",
  "IDPSignout" : "true",
  "RequestSigningAlgorithm" : "rsa-sha256",
  "EncryptedResponses" : "true",
```

```
"IDPInit" : "true"  
}
```

SAML 署名と暗号化

SAML 2.0 サインインは、認証フローのリクエストとレスポンスのベアラーとしてアプリケーションのユーザーを中心に構築されています。転送中にこれらの SAML ドキュメントについてユーザーによる読み取りや変更ができないようにしておきたい場合があります。これを実現するには、ユーザープールの SAML ID プロバイダー (IdP) に SAML 署名と暗号化を追加します。SAML 署名を使用すると、ユーザープールは SAML のサインインリクエストとサインアウトリクエストに署名を追加します。ユーザープールのパブリックキーを使用すると、IdP は、変更されていない SAML リクエストを受信していることを検証できます。次に、IdP が応答し、ユーザーのブラウザセッションに SAML アサーションを渡すと、IdP はそのレスポンスを暗号化して、ユーザーが独自の属性と権限を検査できないようにします。

SAML 署名と暗号化では、ユーザープール SAML オペレーション中のすべての暗号化オペレーションは、Amazon Cognito が生成するユーザープール提供のキーを使用して署名と暗号文を生成する必要があります。現在、外部キーを使用してリクエストに署名することや、暗号化されたアサーションを受け入れるようにユーザープールを設定することはできなくなっています。

Note

ユーザープール証明書は 10 年間有効です。年に 1 回、Amazon Cognito はユーザープールの新しい署名証明書と暗号化証明書を生成します。Amazon Cognito は、署名証明書をリクエストするとき最新の証明書を返し、最新の署名証明書を使用してリクエストに署名します。IdP は、有効期限が切れていないユーザープール暗号化証明書を使用して SAML アサーションを暗号化できます。以前の証明書は、その有効期間の全期間にわたって引き続き有効であり、パブリックキーは、新旧の証明書で変更ありません。ベストプラクティスとしては、プロバイダー設定で証明書を毎年更新します。

トピック

- [IdP からの暗号化された SAML レスポンスの受け入れ](#)
- [SAML リクエストへの署名](#)

IdP からの暗号化された SAML レスポンスの受け入れ

Amazon Cognito と IdP は、ユーザーによるサインインとサインアウトの際に、SAML レスポンスで機密性を確立できます。Amazon Cognito は、パブリック/プライベート RSA キーペアと証明書を、ユーザープールで設定した各外部 SAML プロバイダーに割り当てます。ユーザープール SAML プロバイダーのレスポンス暗号化を有効にする場合は、暗号化された SAML レスポンスをサポートする IdP に証明書をアップロードする必要があります。提供されたキーで IdP がすべての SAML アサーションの暗号化を開始する前は、SAML IdP へのユーザープール接続は機能しません。

以下は、暗号化された SAML サインインのフローの概要です。

1. ユーザーがサインインを開始し、SAML IdP を選択します。
2. ユーザープール [認可エンドポイント](#) は、SAML サインインリクエストを使用してユーザーを SAML IdP にリダイレクトします。ユーザープールは、オプションとして、IdP による整合性の検証を可能にする署名に、このリクエストを添付できます。SAML リクエストに署名する場合は、署名証明書のパブリックキーを使用してユーザープールが署名したリクエストを受け入れるように IdP を設定する必要があります。
3. SAML IdP はユーザーに署名し、SAML レスポンスを生成します。IdP はパブリックキーでレスポンスを暗号化し、ユーザーをユーザープール /saml2/idpresponse エンドポイントにリダイレクトします。IdP は、SAML 2.0 仕様で定義されているようにレスポンスを暗号化する必要があります。詳細については、「[Assertions and Protocols for the OASIS Security Assertion Markup Language \(SAML\) V2.0](#)」の「Element <EncryptedAssertion>」を参照してください。
4. ユーザープールは、SAML レスポンスの暗号文をプライベートキーで復号し、ユーザーに署名します。

Important

ユーザープールで SAML IdP のレスポンス暗号化を有効にする場合、IdP は、プロバイダーに固有のパブリックキーを使用してすべてのレスポンスを暗号化する必要があります。Amazon Cognito は、暗号化をサポートするように設定した SAML 外部 IdP から、暗号化されていない SAML レスポンスを受け入れません。

ユーザープール内の外部 SAML IdP はレスポンス暗号化をサポートでき、各 IdP は独自のキーペアを受け取ります。

AWS マネジメントコンソール

SAML レスポンスの暗号化を設定するには

1. [ユーザープール](#)、[アプリケーションクライアント](#)、および SAML IdP を作成します。
2. SAML ID プロバイダーの作成または編集を行うときは、[リクエストへの署名とレスポンスの暗号化] で、[暗号化された SAML アサーションをこのプロバイダーに要求] というタイトルのチェックボックスをオンにします。
3. ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューで SAML IdP を選択し、[暗号化証明書を表示] を選択します。
4. [.crt としてダウンロード] を選択し、ダウンロードしたファイルを SAML IdP に提供します。証明書内のキーを使用して SAML レスポンスを暗号化するように SAML IdP を設定します。

API/CLI

SAML レスポンスの暗号化を設定するには

[CreateIdentityProvider](#) API リクエストまたは [UpdateIdentityProvider](#) API リクエストの EncryptedResponses パラメータを使用してレスポンス暗号化を設定します。以下は、リクエスト署名をサポートする IdP の ProviderDetails の例です。

```
"ProviderDetails": {
  "MetadataURL" : "https://myidp.example.com/saml/metadata",
  "IDPSignout" : "true",
  "RequestSigningAlgorithm" : "rsa-sha256",
  "EncryptedResponses" : "true",
  "IDPInit" : "true"
}
```

ユーザープールから暗号化証明書を取得するには、[DescribeIdentityProvider](#) API リクエストを行い、レスポンスパラメータ ProviderDetails の ActiveEncryptionCertificate 値を取得します。この証明書を保存し、ユーザープールからのサインインリクエストの暗号化証明書として、この証明書を IdP に提供します。

SAML リクエストへの署名

IdP への SAML 2.0 リクエストの整合性を証明できる機能は、Amazon Cognito の SP が開始した SAML サインインのセキュリティ上のメリットです。ドメインを持つ各ユーザープールは、ユーザー

プール X.509 署名証明書を受け取ります。この証明書内のパブリックキーを使用すると、ユーザープールは、ユーザーが SAML IdP を選択したときにユーザープールが生成するサインアウトリクエストに暗号化署名を適用します。オプションで、SAML サインインリクエストに署名するようにアプリケーションクライアントを設定できます。SAML リクエストに署名すると、IdP は、リクエストの XML メタデータの署名が、指定したユーザープール証明書のパブリックキーと一致することを確認できます。

AWS マネジメントコンソール

SAML リクエストの署名を設定するには

1. [ユーザープール](#)、[アプリケーションクライアント](#)、および SAML IdP を作成します。
2. SAML ID プロバイダーの作成または編集を行うときは、[リクエストへの署名とレスポンスの暗号化] で、[このプロバイダーへの SAML リクエストに署名する] というタイトルのチェックボックスをオンにします。
3. ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] メニューで、[署名証明書を表示] を選択します。
4. [.crt としてダウンロード] を選択し、ダウンロードしたファイルを SAML IdP に提供します。受信 SAML リクエストの署名を検証するように SAML IdP を設定します。

API/CLI

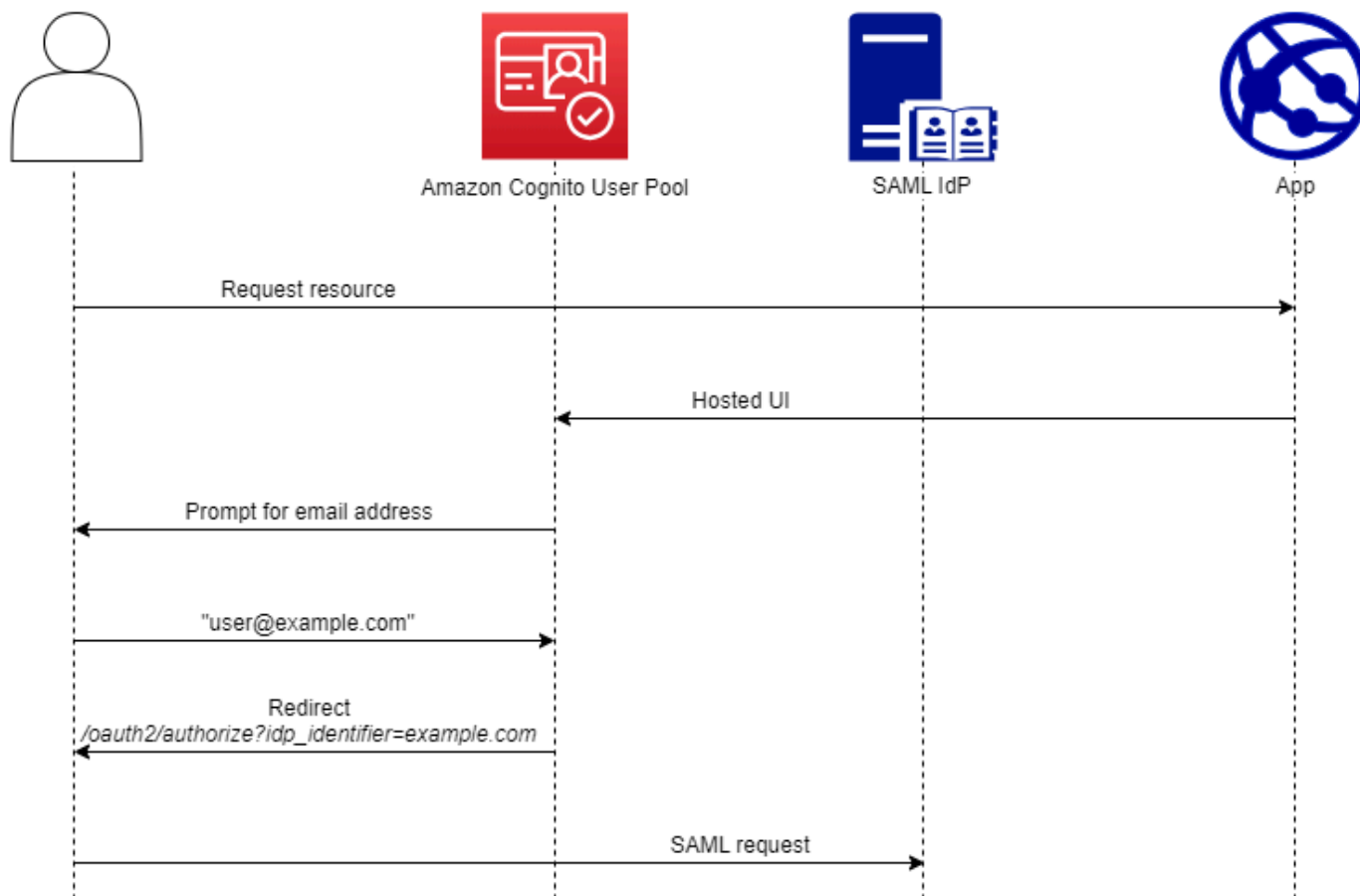
SAML リクエストの署名を設定するには

[CreateIdentityProvider](#) API リクエストまたは [UpdateIdentityProvider](#) API リクエストの `RequestSigningAlgorithm` パラメータを使用してリクエスト署名を設定します。以下は、リクエスト署名をサポートする IdP の `ProviderDetails` の例です。

```
"ProviderDetails": {
  "MetadataURL" : "https://myidp.example.com/saml/metadata",
  "IDPSignout" : "true",
  "RequestSigningAlgorithm" : "rsa-sha256",
  "EncryptedResponses" : "true",
  "IDPInit" : "true"
}
```

SAML ID プロバイダーの名前と識別子

SAML ID プロバイダー (IdP) に名前を付け、IdP 識別子を割り当てると、そのプロバイダーへの、SP が開始したサインインリクエストとサインアウトリクエストのフローを自動化できます。プロバイダーの名前に対する文字列制約の詳細については、[CreateIdentityProvider](#) の `ProviderName` プロパティを参照してください。



SAML プロバイダーの識別子を 50 個まで選択することもできます。識別子は、ユーザープール内の IdP のわかりやすい名前であり、ユーザープール内で一意である必要があります。SAML 識別子がユーザーの E メールドメインと一致すると、マネージドログインは、各ユーザーの E メールアドレスをリクエストし、E メールアドレス内のドメインを評価して、ドメインに対応する IdP にユーザーをリダイレクトします。同じ組織で複数のドメインを所有することができるため、1 つの IdP が複数の識別子を持つことができます。

E メールドメイン識別子の使用の有無にかかわらず、マルチテナントアプリケーションの識別子を使用して、ユーザーを正しい IdP にリダイレクトできます。マネージドログインを完全にバイパスする場合は、ユーザーに提示するリンクをカスタマイズすることで、[認可エンドポイント](#)

[ント](#) を介してユーザーを IdP に直接リダイレクトできます。識別子を使用してユーザーにサインインし、IdP にリダイレクトするには、最初の認可リクエストのリクエストパラメータに `idp_identifier=myidp.example.com` 形式の識別子を含めます。

ユーザーを IdP に渡す別の方法は、IdP の名前を持つパラメータ `identity_provider` を、次の URL 形式で入力することです。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
identity_provider=MySAMLIdP&
client_id=1example23456789&
redirect_uri=https://www.example.com
```

ユーザーが SAML IdP でサインインすると、IdP は、HTTP POST 本文内の SAML レスポンスを使用して `/saml2/idpresponse` エンドポイントにリダイレクトします。Amazon Cognito は SAML アサーションを処理し、レスポンスのクレームが期待を満たしている場合、アプリケーションクライアントのコールバック URL にリダイレクトします。ユーザーがこの方法で認証を完了すれば、ユーザーが IdP とアプリケーションのためにのみウェブページを操作したことになります。

ドメイン形式の IdP 識別子を使用すると、マネージドログインは、サインイン時に E メールアドレスをリクエストし、E メールドメインが IdP 識別子と一致すると、ユーザーを IdP のサインインページにリダイレクトします。例えば、異なる 2 つの会社に所属する従業員によるサインインが必要となるアプリケーションを構築するとします。最初の会社、AnyCompany A は、`exampleA.com` および `exampleA.co.uk` を所有しています。2 番目の会社、AnyCompany B は、`exampleB.com` を所有しています。この例では、以下のように、1 社ごとに 2 つの IdP を設定します。

- IdP A では、識別子 `exampleA.com` および `exampleA.co.uk` を定義します。
- IdP B では、識別子 `exampleB.com` を定義します。

アプリケーションでは、アプリケーションクライアントのマネージドログインを呼び出し、各ユーザーに E メールアドレスの入力を求めます。Amazon Cognito は、E メールアドレスからドメインを取得し、ドメインを IdP とドメイン識別子に関連付け、`idp_identifier` リクエストパラメータを含む [認可エンドポイント](#) へのリクエストを使用して、ユーザーを正しい IdP にリダイレクトします。例えば、ユーザーが `bob@exampleA.co.uk` を入力した場合、ユーザーが操作する次のページは、`https://auth.exampleA.co.uk/sso/saml` の IdP サインインページになります。

また、同じロジックを個別に実装することもできます。アプリケーションでは、ユーザー入力を収集したうえで独自のロジックに従って正しい IdP に関連付けるカスタムフォームを構築できます。ア

アプリケーションテナントごとにカスタムポータルを生成すると、リクエストパラメータ内のテナントの識別子を使用して、各アプリケーションテナントを承認エンドポイントにリンクできます。

E メールアドレスを収集して、マネージドログインのドメインを解析するには、アプリケーションクライアントに割り当てた SAML IdP ごとに 1 つ以上の識別子を割り当てます。デフォルトでは、マネージドログインのサインイン画面に、アプリケーションクライアントに割り当てた IdP ごとにボタンが表示されます。ただし、識別子を正常に割り当てると、クラシックのホストされた UI のサインインページは、次の画像のようになります。

Amazon Cognito のマネージドログインのサインインページに表示される、ローカルユーザーのサインインと、フェデレーションユーザーに E メールアドレスの入力を求めるプロンプト。

Note

クラシックのホストされた UI では、IdP に識別子を割り当てると、アプリケーションクライアントのサインインページで自動的にメールアドレスの入力を求められます。マネージドログインのエクスペリエンスでは、この動作をブランディングエディタで有効にする必要があります。[認証動作] の設定カテゴリで、見出しの [プロバイダーの表示] の下にある [ドメイン検索入力] を選択します。

マネージドログインのドメイン解析では、ドメインを IdP 識別子として使用する必要があります。アプリケーションクライアントの SAML IdP ごとに任意のタイプの識別子を割り当てると、そのアプリケーションのマネージドログインには、IdP の選択ボタンが表示されなくなります。E メール解析やカスタムロジックを使用してリダイレクトを生成する場合は、SAML の IdP 識別子を追加します。サイレントリダイレクトを生成するとともに、マネージドログインに IdP のリストを表示する場合は、識別子を割り当てず、認可リクエストで `identity_provider` リクエストパラメータを使用します。

- アプリケーションクライアントに SAML IdP を 1 つだけ割り当てると、マネージドログインのサインインページには、その IdP でサインインするためのボタンが表示されます。
- アプリケーションクライアントでアクティブ化する SAML IdP ごとに識別子を割り当てると、マネージドログインサインインページには、E メールアドレスの入力をユーザーに求めるプロンプトが表示されます。
- 複数の IdP があり、すべてには識別子を割り当てない場合、マネージドログインのサインインページには、割り当てた IdP ごとにサインインするためのボタンが表示されます。
- IdP に識別子を割り当て、マネージドログインページに特定の IdP ボタンを表示する場合は、新しい IdP を識別子なしでアプリケーションクライアントに追加するか、新しいアプリケーション

クライアントを作成します。既存の IdP を削除し、識別子を用いずにそれを再度追加することもできます。新しい IdP を作成すると、SAML ユーザーが新しいユーザープロフィールを作成します。このようにアクティブなユーザーが重複していると、IdP 設定を変更した月に請求に影響する可能性があります。

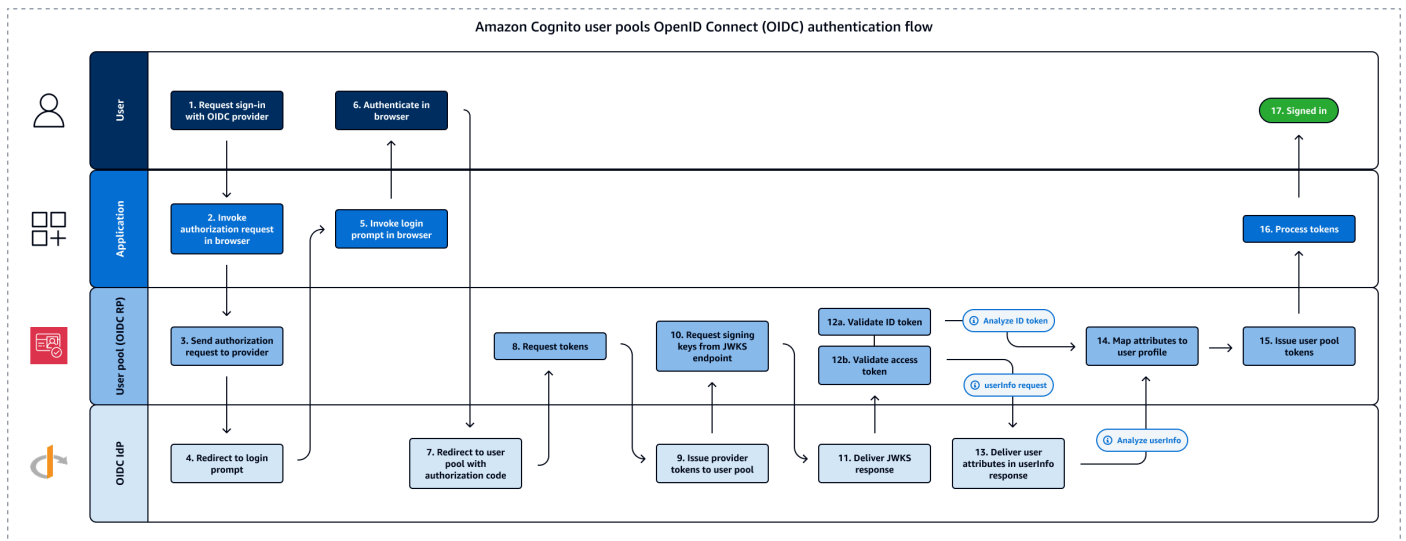
IdP セットアップの詳細については、「[ユーザープールの ID プロバイダーの設定](#)」を参照してください。

ユーザープールでの OIDC ID プロバイダーの使用

ユーザーは、OpenID Connect (OIDC) ID プロバイダー (IdP) の既存アカウントを使用してアプリケーションにサインインできます。OIDC プロバイダーを使用すると、独立したシングルサインオンシステムのユーザーは、アプリケーションがユーザープールの共有形式で OIDC トークンを受信する間、既存の認証情報を提供できます。OIDC IdP を設定するには、ユーザープールを RP として処理するように IdP を設定するとともに、ユーザープールを IdP として処理するようにアプリケーションを設定します。Amazon Cognito は、複数の OIDC IdP とアプリケーションの間の中間ステップとして機能します。ユーザープールは、プロバイダーがユーザープールに直接渡す ID トークンとアクセストークンのクレームに、属性マッピングルールを適用します。次に、Amazon Cognito は、マッピングされたユーザー属性と、[Lambda トリガー](#)を使用して認証フローに加えた追加の調整に基づいて、新しいトークンを発行します。

OIDC IdP でサインインするユーザーは、ユーザープールアプリケーションにアクセスするために、新しい認証情報や情報を提供する必要はありません。アプリケーションは、ユーザープールを使用して、サインインのために IdP にサイレントにリダイレクトできます。この場合、ユーザープールは、アプリケーションのトークン形式を標準化するバックグラウンドのツールとして使用されます。IdP リダイレクトの詳細については、「[認可エンドポイント](#)」を参照してください。

他のサードパーティー ID プロバイダーと同様に、アプリケーションを OIDC プロバイダーに登録し、ユーザープールに接続する IdP アプリケーションに関する情報を取得する必要があります。ユーザープール OIDC IdP には、クライアント ID、クライアントシークレット、リクエストするスコープ、プロバイダーサービスエンドポイントに関する情報が必要です。ユーザープールが検出エンドポイントからプロバイダーの OIDC エンドポイントを検出しますが、ユーザーが手動で入力することもできます。また、プロバイダー ID トークンを調べ、IdP とユーザープール内の属性に関して属性マッピングを作成する必要があります。



この認証フローの詳細については、「[OIDC ユーザープール IdP 認証フロー](#)」を参照してください。

Note

サードパーティー (フェデレーション) 経由のサインインは、Amazon Cognito のユーザープールで使用できます。この機能は、Amazon Cognito アイデンティティプールを使用した OIDC フェデレーションとは無関係です。

OIDC IdP は AWS マネジメントコンソール、、、またはユーザープール API メソッド [CreateIdentityProvider](#) を使用して AWS CLI、ユーザープールに追加できます。

トピック

- [前提条件](#)
- [アプリケーションを OIDC IdP に登録する](#)
- [ユーザープールに OIDC IdP を追加する](#)
- [OIDC IdP の設定をテストする](#)
- [OIDC ユーザープール IdP 認証フロー](#)

前提条件

開始するには、以下が必要です。

- アプリケーションクライアントとユーザープールドメインを使用するユーザープール。詳細については、「[ユーザープールの作成](#)」を参照してください。
- 次の設定を持つ OIDC IdP。
 - `client_secret_post` クライアント認証をサポートします。Amazon Cognito は、IdP の OIDC ディスカバリエンドポイントでの `token_endpoint_auth_methods_supported` クレームをチェックしません。Amazon Cognito は、`client_secret_basic` クライアント認証をサポートしていません。クライアント認証の詳細については、OpenID Connect ドキュメントの「[クライアント認証](#)」を参照してください。
 - `openid_configuration`、`userInfo`、および `JWKS_URI` などの OIDC エンドポイントにのみ HTTPS を使用します。
 - OIDC エンドポイントには TCP ポート 80 および 443 のみを使用します。
 - HMAC-SHA、ECDSA または RSA アルゴリズムで ID トークンにのみ署名します。
 - キー ID `kid` クレームを `JWKS_URI` で発行し、そのトークンに `kid` クレームをフックみます。
 - 有効なルート CA トラストチェーンを持つ、有効期限が切れていないパブリックキーを表示します。

アプリケーションを OIDC IdP に登録する

OIDC IdP をユーザープール設定に追加してアプリケーションクライアントに割り当てる前に、OIDC クライアントアプリケーションを IdP に設定します。ユーザープールは、依拠しているパーティのアプリケーションとして、IdP を使用して認証を管理します。

OIDC IdP に登録する

1. OIDC IdP のデベロッパーアカウントを作成します。

OIDC IdP へのリンク

OIDC IdP	インストール方法	OIDC 検出 URL
Salesforce	OpenID Connect ID プロバイダーとしての Salesforce	<code>https://MyDomainName.my.salesforce.com/.well-known/openid-configuration</code>
OneLogin	OIDC 対応アプリケーションを接続する	<code>https://your-domain.onelogin.com /</code>

OIDC IdP	インストール方法	OIDC 検出 URL
		oidc/2/.well-known/openid-configuration
JumpCloud	OIDC による SSO	https://oauth.id.jumpcloud.com/.well-known/openid-configuration
Okta	Okta ID プロバイダーのインストール	https:// <i>Your Okta subdomain</i> .okta.com/.well-known/openid-configuration
Microsoft Entra ID	Microsoft ID プラットフォームでの OpenID Connect	https://login.microsoftonline.com/ <i>{tenant}</i> /v2.0 tenant の値には、テナント ID、common、organizations、または consumers を含めることができます。

- OIDC IdP を使用して、/oauth2/idpresponse エンドポイントを持つユーザープールドメインを登録します。そうすることで、OIDC IdP が後ほど、ユーザーの認証時に Amazon Cognito からこれを受け入れることが確実にになります。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
```

- ユーザーディレクトリとユーザープールで共有する[スコープ](#)を選択します。OIDC IdP がユーザー情報を提供するには、スコープとして openid が必要です。email スコープは、email および email_verified の[クレーム](#)へのアクセスを付与するために必要です。OIDC 仕様のその他のスコープは、すべてのユーザー属性のための profile と、phone_number および phone_number_verified のための phone です。
- OIDC IdP は、クライアント ID とクライアントシークレットを提供します。これらの値をメモし、後でユーザープールに追加する OIDC IdP の設定に追加します。

例: Salesforce を OIDC IdP としてユーザープールで使用する

OIDC 互換 IdP (Salesforce など) とユーザープールの間で信頼性を確立するときに OIDC IdP を使用します。

1. Salesforce 開発者ウェブサイトでは[アカウントを作成](#)します。
2. 前のステップで設定した開発者アカウントを使用して[サインイン](#)します。
3. Salesforce ページで、次のいずれかの操作を行います。
 - Lightning Experience を使用している場合は、歯車アイコンを選択してから、[Setup Home] (ホームの設定) を選択します。
 - Salesforce Classic を使用しており、ユーザーインターフェイスのヘッダーに [Setup (設定)] が表示されている場合は、選択します。
 - Salesforce Classic を使用しており、ヘッダーに [Setup (設定)] が表示されていない場合は、上部のナビゲーションバーで名前を選択し、ドロップダウンリストより [Setup (設定)] を選択します。
4. 左のナビゲーションバーで、[Company Settings (組織の設定)] を選択します。
5. ナビゲーションバーで、[Domain] (ドメイン) を選択してドメインを入力し、[Create] (作成) を入力します。
6. 左のナビゲーションバーで、[Platform Tools] (プラットフォームツール) に移動し、[Apps] (アプリケーション) を選択します。
7. [アプリケーションマネージャ] を選択します。
8. a. [New connected app] (新規接続アプリケーション) を選択します。
b. 必須フィールドに入力します。

[Start URL] (開始 URL) で、Salesforce IdP でサインインするユーザープールドメインの /authorize エンドポイントの URL を入力します。ユーザーが接続アプリケーションにアクセスすると、Salesforce はこの URL に誘導してサインインを完了します。次に、Salesforce はユーザーをアプリクライアントに関連付けたコールバック URL にリダイレクトします。

```
https://mydomain.auth.us-east-1.amazoncognito.com/authorize?
response_type=code&client_id=<your_client_id>&redirect_uri=https://
www.example.com&identity_provider=CorpSalesforce
```

- c. OAuth 設定を有効にし、コールバック URL にユーザープールドメインの /oauth2/idpresponse エンドポイントの URL を入力します。これは、Amazon Cognito が OAuth トークンと交換する認可コードを Salesforce が発行する URL です。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
```

9. [スコープ](#)を選択します。スコープ openid を含める必要があります。email および email_verified の [クレーム](#)へのアクセスを許可するには、email スコープを追加します。スコープはスペースで区切ります。
10. [作成] を選択します。

Salesforce では、クライアント ID は [コンシューマーキー]、クライアントシークレットは [コンシューマーシークレット] と呼ばれます。クライアント ID とクライアントシークレットを書き留めます。これらは次のセクションで使用します。

ユーザープールに OIDC IdP を追加する

IdP を設定したら、OIDC IdP で認証リクエストを処理するようにユーザープールを設定できます。

Amazon Cognito console

コンソールで OIDC IdP を追加する

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. ナビゲーションメニューから [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択し、[ID プロバイダーを追加] を選択します。
5. [OpenID Connect] IdP を選択します。
6. [プロバイダー名] に一意の名前を入力します。
7. IdP の [クライアント ID] を入力します。これは、OIDC IdP で構築するアプリケーションクライアントの ID です。指定するクライアント ID は、https://[*your user pool domain*]/oauth2/idpresponse のコールバック URL で設定した OIDC プロバイダーである必要があります。

8. IdP の [クライアントシークレット] を入力します。これは、前のステップで入力したのと同じアプリケーションクライアントのクライアントシークレットである必要があります。
9. このプロバイダーの [Authorized scopes] (承認済みスコープ) を入力します。スコープは、アプリケーションがプロバイダーにリクエストするユーザー属性のグループ (name および email など) を定義します。[OAuth 2.0](#) 仕様に従い、スコープはスペースで区切る必要があります。

IdP は、ユーザーがサインインするときに、これらの属性をアプリケーションに提供することと同意するようユーザーに求める場合があります。

10. [属性のリクエストメソッド] を選択します。IdP は、userInfo エンドポイントへのリクエストを GET または POST の形式にするよう求める場合があります。例えば、Amazon Cognito の userInfo エンドポイントには HTTP GET リクエストが必要です。
11. [セットアップ方法] で、IdP の主要な OIDC フェデレーションエンドポイントへのパスを、ユーザープールで決定する方法を選択します。通常、IdP は、発行者のベース URL で /well-known/openid-configuration エンドポイントをホストします。これが該当するプロバイダーの場合、[発行者 URL を通じた自動入力] オプションは、そのベース URL の入力を求め、そこから /well-known/openid-configuration パスへのアクセスを試み、表示されたエンドポイントを読み取ります。エンドポイントパスが通常とは異なっていたり、代替プロキシを介して 1 つ以上のエンドポイントにリクエストを渡したりする場合があります。その場合は、[手動入力] を選択し、authorization、token、userInfo、jwks_uri エンドポイントのパスを指定します。

 Note

URL は、https:// で始める必要があり、末尾にスラッシュ (/) を使用することはできません。この URL では、ポート番号 443 および 80 のみを使用できます。例えば、Salesforce では次の URL を使用します。

https://login.salesforce.com

自動入力を選択した場合、検出ドキュメントは

authorization_endpoint、token_endpoint、userinfo_endpoint、および jwks_uri の値に HTTPS を使用する必要があります。そうしない場合は、ログインが失敗します。

12. [OpenID Connect プロバイダーとユーザープール間の属性をマッピングする] で属性マッピングルールを設定します。ユーザープール属性は Amazon Cognito ユーザープロフィールの送信先属性であり、OpenID Connect 属性は ID トークンのクレームまたは userInfo レ

スポンスで Amazon Cognito が検索するソース属性です。Amazon Cognito は、OIDC クレーム sub を送信先ユーザープロファイルの username に自動的にマッピングします。

詳細については、「[IdP 属性をプロファイルとトークンにマッピングする](#)」を参照してください。

13. [ID プロバイダーを追加] を選択します。
14. [アプリケーションクライアント] メニューで、リストからアプリケーションクライアントを選択します。[ログインページ] タブに移動し、[マネージドログインページの設定] で [編集] を選択します。[ID プロバイダー] を見つけ、新しい OIDC IdP を追加します。
15. [Save changes] (変更の保存) をクリックします。

API/CLI

[CreateIdentityProvider](#) の例 2 の OIDC 設定を参照してください。この構文を変更して、CreateIdentityProvider と UpdateIdentityProvider のリクエスト本文として使用するか、[create-identity-provider](#) の --cli-input-json 入力ファイルとして使用できます。

OIDC IdP の設定をテストする

アプリケーションでは、ユーザーが OIDC プロバイダーでサインインできるように、ユーザーのクライアントでブラウザを起動する必要があります。前のセクションのセットアップ手順を完了したら、プロバイダーを使用したサインインをテストします。次の URL の例では、プレフィックスドメインを持つユーザープールのサインインページをロードします。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://www.example.com
```

このリンクは、[アプリケーションクライアント] メニューに移動してアプリケーションクライアントを選択し、[ログインページ] タブに移動して [ログインページを表示] を選択したときに、Amazon Cognito に表示されるページです。ユーザープールドメインの詳細については、「[ユーザープールのドメインを設定する](#)」を参照してください。クライアント ID とコールバック URL など、アプリケーションクライアントの詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

次のリンク例では、identity_provider クエリパラメータを使用して、[認可エンドポイント](#) から MyOIDCIdP プロバイダーへのサイレントリダイレクトを設定します。この URL は、マネージドロ

グインによるインタラクティブなユーザープールのサインインをバイパスし、IdP サインインページに直接移動します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
identity_provider=MyOIDCIdP&response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com
```

OIDC ユーザープール IdP 認証フロー

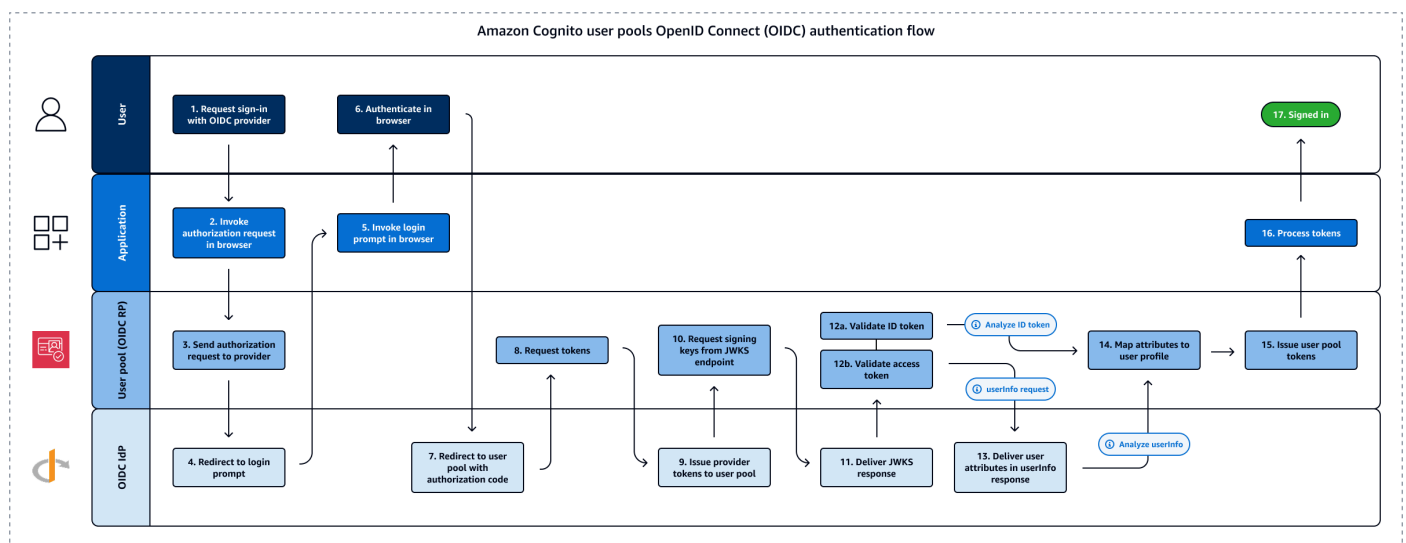
OpenID Connect (OIDC) サインインを使用すると、ユーザープールは ID プロバイダー (IdP) による認可コードサインインフローを自動化します。ユーザーが IdP でサインインを完了すると、Amazon Cognito は外部プロバイダーの `oauth2/idpresponse` エンドポイントでコードを収集します。結果として得られたアクセストークンを使用して、ユーザープールは、IdP `userInfo` エンドポイントにクエリを実行してユーザー属性を取得します。その後、ユーザープールは、受信した属性を、設定済みの属性マッピングルールと比較し、それに応じてユーザーのプロファイルと ID トークンを入力します。

OIDC プロバイダー設定でリクエストする OAuth 2.0 スコープは、IdP が Amazon Cognito に提供するユーザー属性を定義します。セキュリティのベストプラクティスとしては、ユーザープールにマッピングする属性に対応するスコープのみをリクエストします。例えば、ユーザープールが `openid profile` をリクエストすると、考えられる属性がすべて取得されますが、`openid email phone_number` をリクエストすると、ユーザーの E メールアドレスと電話番号のみが取得されます。[OIDC IdP にリクエスト](#)するスコープについては、[アプリケーションクライアント](#)とユーザープール認証リクエストで認可とリクエストを行うスコープとは異なるものとして設定できます。

ユーザーが OIDC IdP を使用してアプリケーションにサインインする場合、ユーザープールは、以下の認証フローを実行します。

1. ユーザーはマネージドログインのサインインページにアクセスし、OIDC IdP でサインインすることを選択します。
2. アプリケーションは、ユーザーのブラウザを、ユーザープールの承認エンドポイントに誘導します。
3. ユーザープールは、OIDC IdP の承認エンドポイントにリクエストをリダイレクトします。
4. IdP にログインプロンプトが表示されます。
5. アプリケーションでは、ユーザーのセッションに OIDC IdP のサインインプロンプトが表示されます。
6. ユーザーは IdP の認証情報を入力するか、認証済みセッションの Cookie を提示します。

7. ユーザーが認証を行うと、OIDC IdP が認可コードを使用して Amazon Cognito にリダイレクトします。
8. ユーザープールは、認証コードを ID トークンおよびアクセストークンと交換します。IdP にスコープ `openid` を設定すると、Amazon Cognito はアクセストークンを受け取ります。ID トークンのクレームと `userInfo` レスポンスは、IdP 設定のその他のスコープ (`profile` や `email` など) によって決まります。
9. IdP はリクエストされたトークンを発行します。
10. ユーザープールは、IdP 設定の発行者 URL から IdP の `jwks_uri` エンドポイントへのパスを決定し、JSON ウェブキーセット (JWKS) エンドポイントに対してトークン署名キーをリクエストします。
11. IdP は JWKS エンドポイントから署名キーを返します。
12. ユーザープールは、トークン内の署名と有効期限のデータから IdP トークンを検証します。
13. ユーザープールは、アクセストークンを使用して IdP の `userInfo` エンドポイントへのリクエストを承認します。IdP は、アクセストークンのスコープに基づいて、ユーザーデータを返します。
14. ユーザープールは、IdP からの ID トークンと `userInfo` レスポンスを、ユーザープール内の属性マッピングルールと比較します。マッピングされた IdP 属性をユーザープールのプロフィール属性に書き込みます。
15. Amazon Cognito がアプリケーションのベアラートークンを発行します。これには ID トークン、アクセストークン、および更新トークンが含まれる場合があります。
16. アプリケーションは、ユーザープールのトークンを処理し、ユーザーをサインインさせます。



Note

Amazon Cognito は 5 分以内に完了しない認証リクエストをキャンセルし、ユーザーをマネージドログインにリダイレクトします。ページには、Something went wrong というエラーメッセージが表示されます。

OIDC は、OAuth 2.0 の上に位置する ID レイヤーです。OAuth 2.0 は、IdP から OIDC クライアントアプリ (証明書利用者) に発行される JSON 形式 (JWT) の ID トークンを指定します。Amazon Cognito の OIDC Relying Party としての追加に関する情報は、OIDC IdP のドキュメントを参照してください。

ユーザーが認可コード付与を使用して認証すると、ユーザープールは ID、アクセス、更新トークンを返します。ID トークンは ID 管理用の [OIDC](#) 標準トークンです。アクセストークンは [OAuth 2.0](#) 標準トークンです。ユーザープールのアプリケーションクライアントがサポートできる付与タイプの詳細については、「[認可エンドポイント](#)」を参照してください。

ユーザープールが OIDC プロバイダーからのクレームを処理する方法

ユーザーがサードパーティの OIDC プロバイダーでサインインを完了すると、マネージドログインは IdP から認証コードを取得します。ユーザープールは、アクセストークンと ID トークンの認可コードを IdP の token エンドポイントと交換します。ユーザープールはこれらのトークンをユーザーやアプリに渡すのではなく、これらを使って独自のトークンのクレームで提示するデータを使用してユーザープロフィールを構築します。

Amazon Cognito はアクセストークンを個別に検証しません。代わりに、userInfoプロバイダーのエンドポイントにユーザー属性情報をリクエストし、トークンが有効でない場合はリクエストが拒否されることを想定しています。

Amazon Cognito は、以下のチェックを行ってプロバイダー ID トークンを検証します。

1. プロバイダーが RSA、HMAC、Elliptic Curve というセットのアルゴリズムを使用してトークンに署名したことをチェックします。
2. プロバイダーが非対称署名アルゴリズムを使用してトークンに署名した場合は、トークンの kid クレームの署名キー ID がプロバイダーの jwks_uri エンドポイントに表示されていることを確認します。Amazon Cognito は、処理する IdP ID トークンごとに、IdP 設定の JWKS エンドポイントから署名キーを更新します。
3. ID トークンの署名を、プロバイダーのメタデータに基づいて想定される署名と比較します。

4. iss クレームを IdP に設定された OIDC 発行者と比較します。
5. aud クレームが IdP で設定されているクライアント ID と一致するか、または aud クレームに複数の値がある場合は設定されたクライアント ID が含まれているかを比較します。
6. exp クレームのタイムスタンプが現在の時刻より前でないことをチェックします。

ユーザープールは ID トークンを検証し、プロバイダーアクセストークンを使用してプロバイダーの `userInfo` エンドポイントへのリクエストを試みます。アクセストークンのスコープによって読み取りが許可されたユーザープロフィール情報がすべて取得されます。次にユーザープールはユーザープールの要求に従って設定したユーザー属性を検索します。プロバイダー設定で、必要な属性の属性マッピングを作成する必要があります。ユーザープールはプロバイダー ID トークンと `userInfo` レスポンスをチェックします。ユーザープールは、マッピングルールに一致するすべてのクレームをユーザープールのユーザープロフィールのユーザー属性に書き込みます。ユーザープールは、マッピングルールに一致するが必須ではなく、プロバイダーのクレームにも含まれていない属性を無視します。

IdP 属性をプロフィールとトークンにマッピングする

Amazon Cognito を含む ID プロバイダー (IdP) サービスは、通常、ユーザーに関する詳細情報を記録できます。ユーザーの勤務先の会社、連絡方法、その他の識別情報を把握しておきたい場合があります。しかしながら、これらの属性の形式はプロバイダーによって違いがあります。例えば、ユーザープールを使用して 3 つの異なるベンダーから 3 つの IdP をセットアップし、各ベンダーの SAML アサーション、ID トークン、`userInfo` ペイロードの例を調べるとします。ユーザーの E メールアドレスの表記は、ある IdP では `email`、別の IdP では `emailaddress`、もう 1 つの IdP では `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress` となります。

IdP ユーザープールと統合することで得られる主なメリットは、さまざまな属性名を、一貫性のある予測可能な共有属性名を持つ単一の OIDC トークンスキーマにマッピングできることです。これにより、デベロッパーは複雑なさまざまなシングルサインオンイベントを処理するためのロジックを維持する必要がなくなります。こうした形式の統合が属性マッピングです。ユーザープール属性マッピングは、IdP 属性名を、対応するユーザープール属性名に割り当てます。例えば、`emailaddress` クレームの値を標準ユーザープール属性 `email` に書き込むようにユーザープールを設定できます。

各ユーザープール IdP には、個別の属性マッピングスキーマがあります。IdP の属性マッピングを指定するには、Amazon Cognito コンソール、AWS SDK、またはユーザープール REST API でユーザープール ID プロバイダーを設定します。

マッピングについて知っておくべきこと

ユーザー属性マッピングの設定を開始する前に、以下の重要な詳細事項を確認します。

- フェデレーションユーザーがアプリケーションにサインインする際、マッピングは、ユーザープールが必要とする各ユーザープール属性に存在している必要があります。例えば、サインアップするためにユーザープールで email 属性が必要となる場合には、この属性を IdP から該当するものにマッピングします。
- デフォルトで、マップされた E メールアドレスは未検証です。ワンタイムコードを使用してマップされた E メールアドレスを検証することはできません。その代わりに IdP からの属性をマップして、検証ステータスを取得します。例えば、Google とほとんどの OIDC プロバイダーには email_verified 属性が含まれています。
- ID プロバイダー (IdP) のトークンをユーザープールのカスタム属性にマッピングできます。ソーシャルプロバイダーはアクセストークンを提示し、OIDC プロバイダーはアクセストークンと ID トークンを提示します。トークンをマッピングするには、最大長さ 2,048 文字のカスタム属性を追加し、アプリケーションにその属性への書き込みアクセス許可を付与し、IdP からの access_token または id_token をカスタム属性にマッピングします。
- マッピングされた各ユーザープールの属性では、値の最大長さである 2048 文字が、Amazon Cognito が IdP から取得する値に対して十分な長さである必要があります。そうではない場合は、ユーザーがアプリケーションにサインインするときに Amazon Cognito がエラーを報告します。Amazon Cognito は、トークンの長さが 2,048 文字を超える場合、IdP トークンのカスタム属性へのマッピングをサポートしていません。
- Amazon Cognito は、フェデレーション IdP がパスする特定のクレームから、フェデレーションユーザーのプロファイル内の username 属性を、次の表に示すように派生します。Amazon Cognito は、この属性値の先頭に、MyOIDCIdP_[sub] などの IdP の名前を追加します。フェデレーションユーザーについて、外部ユーザーディレクトリ内の属性と完全に一致する属性を持たせる場合は、その属性を preferred_username などの Amazon Cognito サインイン属性にマッピングします。

ID プロバイダー	username ソース属性
Facebook	id
Google	sub
Login with Amazon	user_id

ID プロバイダー	username ソース属性
Apple でのサインイン	sub
SAML プロバイダー	NameID
OpenID Connect (OIDC) プロバイダー	sub

- ユーザープールが [大文字と小文字を区別しない](#)ものである場合、Amazon Cognito は、フェデレーションユーザーの自動生成されたユーザー名におけるユーザー名ソース属性を小文字に変換します。大文字と小文字を区別するユーザープールのユーザー名の例は次のとおりです: MySAML_TestUser@example.com。大文字と小文字を区別しないユーザープールでは同じユーザー名が次のようになります: MySAML_testuser@example.com。

大文字と小文字を区別しないユーザープールでは、ユーザー名を処理する Lambda トリガーは、ユーザー名ソース属性について大文字と小文字が混在するクレームに対して、この変更を考慮する必要があります。IdP を、現在のユーザープールとは異なる大文字と小文字の区別設定を持つユーザープールにリンクするには、新しいユーザープールを作成します。

- Amazon Cognito は、ユーザーがアプリケーションにサインインするときにマップされたユーザープール属性を更新できる必要があります。ユーザーが IdP 経由でサインインすると、Amazon Cognito は IdP からの最新情報でマップされた属性を更新します。Amazon Cognito は、マップされた属性の現行の値が最新情報とすでに一致している場合でも、各属性を更新します。Amazon Cognito が属性を更新できることを確実にするため、以下の要件をチェックしてください。
- IdP からマッピングするユーザープールのカスタム属性は、すべて変更可能である必要があります。変更可能なカスタム属性はいつでも更新できます。一方、ユーザーのイミュータブル (変更不可) カスタム属性は、ユーザープロフィールの初回作成時を除き、値を設定できません。Amazon Cognito コンソールで変更可能なカスタム属性を作成するには、[サインアップエクスペリエンス] メニューの [カスタム属性を追加] を選択して追加した属性の [変更可能] チェックボックスをオンにします。または、[CreateUserPool](#) API オペレーションを使用してユーザープールを作成した場合、これらの各属性の Mutable パラメータを true に設定できます。IdP が、マッピングされたイミュータブルな属性の値を送信すると、Amazon Cognito はエラーを返し、サインインは失敗します。
- アプリケーションのアプリクライアントの設定では、マッピングされた属性を書き込み可能にする必要があります。書き込み可能な属性は、Amazon Cognito コンソールの [App clients] (アプリクライアント) ページで設定できます。また、[CreateUserPoolClient](#) API を使用してアプリクライアントを作成する場合、これらの属性を WriteAttributes 配列に追加できます。IdP

が、マッピングされた書き込み不可の属性の値を送信する場合、Amazon Cognito は属性値を設定せず、認証に進みます。

- IdP 属性に複数の値が含まれている場合、Amazon Cognito はすべての値を角括弧文字 [および] で囲まれた 1 つのカンマ区切り文字列にフラット化します。Amazon Cognito は、.、-、*、_ を除く、英数字以外の文字を含む値を URL フォームエンコードします。値は、アプリでの使用前にデコードおよび解析しておく必要があります。
- 送信先属性は、サインインまたは管理アクションによって変更されない限り、属性マッピングルールによって割り当てられた値を保持します。プロバイダトークンまたは SAML アサーションでソース属性が送信されなくなった場合でも、Amazon Cognito はユーザーから属性を削除しません。以下のアクションでは、フェデレーションユーザーのユーザープールプロファイルから属性の値を削除します。
 1. IdP はソース属性に空白の値を送信し、マッピングルールによって空白の値を送信先属性に適用します。
 2. [DeleteUserAttributes](#) リクエストまたは [AdminDeleteUserAttributes](#) リクエストを使用して、マッピングされた属性の値をクリアします。

ユーザープールの ID プロバイダー属性マッピングを指定する (AWS マネジメントコンソール)

を使用して AWS マネジメントコンソール、ユーザープールの IdP の属性マッピングを指定できます。

Note

Amazon Cognito は、着信トークンに着信クレームが存在する場合に限り、そのクレームをユーザープール属性にマップします。以前にマッピングされたクレームが着信トークンに存在しなくなった場合、そのクレームは削除または変更されません。削除されたクレームのマッピングがアプリケーションで必要な場合は、事前認証 Lambda トリガーを使用して認証中にカスタム属性を削除し、それらの属性を着信トークンから再入力できます。

ソーシャル IdP 属性マッピングを指定するには

1. [Amazon Cognito コンソール](#) にサインインします。プロンプトが表示されたら、AWS 認証情報を入力します。

2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択します。
4. [ID プロバイダーを追加] を選択するか、設定した [Facebook]、[Google]、[Amazon]、または [Apple] の IdP を選択します。[Attribute mapping] (属性マッピング) を検索し、[Edit] (編集) を選択します。

ソーシャル IdP の追加の詳細については、「[ユーザープールによるソーシャル ID プロバイダーの使用](#)」を参照してください。

5. マッピングする必要がある各属性について、以下のステップを完了してください。
 - a. [User pool attribute] (ユーザープール属性) 列から属性を選択します。これは、ユーザープール内のユーザープロファイルに割り当てられる属性です。カスタム属性は、標準属性の後にリストされます。
 - b. [**<provider>** attribute] (<プロバイダー> 属性) 列から属性を選択します。これは、プロバイダーディレクトリから渡される属性になります。ソーシャルプロバイダーの既知の属性は、ドロップダウンリストに表示されます。
 - c. IdP と Amazon Cognito の間に追加の属性をマッピングするには、[Add another attribute] (その他の属性を追加) を選択します。
6. [Save changes] (変更の保存) をクリックします。

SAML プロバイダー属性マッピングを指定する

1. [Amazon Cognito コンソール](#)にサインインします。プロンプトが表示されたら、AWS 認証情報を入力します。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [ソーシャルプロバイダーと外部プロバイダー] メニューを選択します。
4. [Add an identity provider] (ID プロバイダーの追加) を選択するか、設定した SAML IdP を選択します。[Attribute mapping] (属性マッピング) を探し、[Edit] (編集) を選択します。SAML IdP の詳細については、「[ユーザープールによる SAML ID プロバイダーの使用](#)」を参照してください。
5. マッピングする必要がある各属性について、以下のステップを完了してください。

- a. [User pool attribute] (ユーザープール属性) 列から属性を選択します。これは、ユーザープール内のユーザープロフィールに割り当てられる属性です。カスタム属性は、標準属性の後にリストされます。
- b. [SAML attribute] (SAML 属性) 列から属性を選択します。これは、プロバイダーディレクトリから渡される属性になります。

IdP は、参考として SAML アサーションのサンプルを提供する場合があります。IdP の中には、email などの単純な名前を使用するものもあれば、次のような URL 形式の属性名を使用するものもあります。

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

- c. IdP と Amazon Cognito の間に追加の属性をマッピングするには、[Add another attribute] (その他の属性を追加) を選択します。
6. [Save changes] (変更の保存) をクリックします。

ユーザープール (AWS CLI および AWS API) の ID プロバイダー属性マッピングの指定

次の [CreateIdentityProvider](#) または [UpdateIdentityProvider](#) のリクエスト本文は、SAML プロバイダーの「MyIdP」属性 emailaddress、birthdate、phone を、ユーザープール属性 email、birthdate、phone_number に (この順序で) マッピングします。これは SAML 2.0 プロバイダーの完全なリクエスト本文です。リクエスト本文は IdP タイプと特定の詳細によって異なります。属性マッピングは AttributeMapping パラメータにあります。

```
{
  "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"
}
```

以下のコマンドを使用して、ユーザープールの IdP 属性マッピングを指定します。

プロバイダーの作成時に属性マッピングを指定する

- AWS CLI: `aws cognito-idp create-identity-provider`

メタデータファイルの例: `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`

ここでは `details.json` に以下が含まれます。

```
{
  "MetadataFile": "<SAML metadata XML>"
}
```

Note

`<SAML ##### XML>` に引用符が (") 含まれる場合は、エスケープ (\") する必要があります。

メタデータ URL の例:

```
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](#)

既存の IdP の属性マッピングを指定するには

- AWS CLI: `aws cognito-idp update-identity-provider`

```
例: 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](#)

特定の IdP の属性マッピングに関する情報を取得するには

- AWS CLI: `aws cognito-idp describe-identity-provider`

```
例: aws cognito-idp describe-identity-provider --user-pool-id
<user_pool_id> --provider-name <provider_name>
```

- API/SDK: [DescribeIdentityProvider](#)

フェデレーションユーザーを既存のユーザープロフィールにリンクする

多くの場合、同じユーザーが、ユーザープールに接続している複数の ID プロバイダー (IdP) にプロフィールを持っています。Amazon Cognito では、ユーザーが出現するたびにディレクトリ内の同じユーザープロフィールにリンクできます。このようにして、複数の IdP ユーザーを持つ 1 人のユーザーが、アプリで一貫したエクスペリエンスを持つことができます。[AdminLinkProviderForUser](#) は、フェデレーションディレクトリ内のユーザーの一意の ID をユーザープール内のユーザーとして認識するように Amazon Cognito に指示します。ユーザープール内のユーザーは、ユーザープロフィールに関連付けられているフェデレーション ID が 0 個以上ある場合、[課金](#)の目的で 1 人の月間アクティブユーザー (MAU) としてカウントされます。

フェデレーションユーザーが初めてユーザープールにサインインすると、Amazon Cognito はその ID にリンクされたローカルプロフィールを探します。リンクされたプロフィールが存在しない場合、ユーザープールは新しいプロフィールを作成します。ローカルプロフィールを作成して、AdminLinkProviderForUser API リクエストで、計画されたプリステージタスクまたは[サインアップ前の Lambda トリガー](#) のいずれかで初めてサインインする前に、フェデレーション

ユーザーにいつでもリンクできます。ユーザーがサインインし、Amazon Cognito がリンクされたローカルプロファイルを検出すると、ユーザープールはユーザーのクレームを読み取り、IdP のマッピングルールと比較します。次に、ユーザープールは、ログイン時にマッピングされたクレームを反映して、リンクされたローカルプロファイルを更新します。これにより、アクセスクレームを使用してローカルプロファイルを設定し、ID クレームをプロバイダーとの間で最新の状態に保つことができます。Amazon Cognito により、フェデレーションユーザーとリンクされたプロファイルがマッチングされると、フェデレーションユーザーは常にそのプロファイルにサインインします。次に、ユーザーの他のプロバイダー ID を同じプロファイルにリンクして、ユーザーのアプリエクスペリエンスを一貫させることができます。以前にサインインしたフェデレーションユーザーをリンクするには、まず既存のプロファイルを削除する必要があります。既存のプロファイルは、`[Provider name]_identifier` という形式で識別できます。例えば、`LoginWithAmazon_amzn1.account.AFAEXAMPLE`。ユーザーを作成してサードパーティーユーザー ID にリンクした場合、そのユーザーは、作成時に付けられたユーザー名と、リンクされた ID の詳細を含む `identities` 属性を持つことになります。

Important

`AdminLinkProviderForUser` は、外部フェデレーション ID を持つユーザーが、ユーザープール内の既存のユーザーとしてサインインすることを許可するため、アプリケーション所有者によって信頼されている外部 IdP およびプロバイダー属性でのみ使用されることが重要です。

例えば、複数のお客様と共有するアプリを使用するマネージドサービスプロバイダー (MSP) の場合です。お客様はそれぞれ、Active Directory Federation Services (ADFS) を使用してアプリにサインインします。IT 管理者 Carlos は、各お客様のドメインにアカウントを持っています。IdP に関係なく、Carlos がサインインするたびにアプリ管理者として認識されるようにします。

ADFS IdP は、Carlos の SAML アサーションの email クレームで Carlos の E メールアドレス `mvp_carlos@example.com` を Amazon Cognito に提示します。ユーザー名 Carlos を使用してユーザープールにユーザーを作成します。以下の AWS Command Line Interface (AWS CLI) コマンドは、IdPs ADFS1、ADFS2、ADFS3 から Carlos の ID をリンクします。

Note

特定の属性クレームに基づいてユーザーをリンクできます。この機能は OIDC と SAML IdP に固有のものです。他のプロバイダータイプでは、固定ソース属性に基づいてリンクする必要があります。詳細については、「[AdminLinkProviderForUser](#)」を参照してください。

ソーシャル IdP をユーザープロフィールにリンクするとき、ProviderAttributeName に Cognito_Subject を設定する必要があります。ProviderAttributeValue は、IdP のユーザーの一意的識別子でなければなりません。

```
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
```

ユーザープールのユーザープロフィール Carlos は、次の identities 属性を持つようになりました。

```
[{  
  "userId": "msp_carlos@example.com",  
  "providerName": "ADFS1",  
  "providerType": "SAML",  
  "issuer": "http://auth.example.com",  
  "primary": false,  
  "dateCreated": 111111111111111  
}, {  
  "userId": "msp_carlos@example.com",  
  "providerName": "ADFS2",  
  "providerType": "SAML",  
  "issuer": "http://auth2.example.com",  
  "primary": false,  
  "dateCreated": 111111111111111  
}, {  
  "userId": "msp_carlos@example.com",
```

```
"providerName": "ADFS3",
"providerType": "SAML",
"issuer": "http://auth3.example.com",
"primary": false,
"dateCreated": 1111111111111111
}]
```

フェデレーションユーザーのリンクについて知っておくべきこと

- 各ユーザープロファイルに最大 5 人のフェデレーションユーザーをリンクできます。
- `AdminLinkProviderForUser` API リクエストにおける `SourceUser` の `ProviderAttributeName` パラメータで定義されているように、最大 5 つの IdP 属性クレームから各 IdP にユーザーをリンクできます。例えば、少なくとも 1 人のユーザーをソース属性 `email`、`phone`、`department`、`given_name`、`location` にリンクしている場合、これらの 5 つの属性のいずれかにおいてのみ追加ユーザーをリンクできます。
- フェデレーションユーザーは、既存のフェデレーションユーザープロファイルまたはローカルユーザーにリンクできます。
- プロバイダーを のユーザープロファイルにリンクすることはできません AWS マネジメントコンソール。
- ユーザーの ID トークンには、`identities` クレームに関連するすべてのプロバイダーが含まれます。
- [AdminSetUserPassword](#) API リクエストで、自動的に作成されたフェデレーションユーザープロファイルのパスワードを設定できます。その後、そのユーザーのステータスは `EXTERNAL_PROVIDER` から `CONFIRMED` に変わります。この状態のユーザーは、フェデレーションユーザーとしてサインインし、リンクされたローカルユーザーのように API で認証フローを開始できます。また、[ChangePassword](#) や [UpdateUserAttributes](#) などのトークン認証された API リクエストでパスワードや属性を変更することもできます。セキュリティのベストプラクティスとして、またユーザーを外部 IdP と同期した状態を維持するため、フェデレーションユーザープロファイルにパスワードを設定しないでください。代わりに、`AdminLinkProviderForUser` でユーザーをローカルプロファイルにリンクしてください。
- Amazon Cognito は、ユーザーが IdP からサインインするときに、リンクされたローカルユーザープロファイルにユーザー属性を入力します。Amazon Cognito は OIDC IdP からの ID トークン内のアイデンティティ要求を処理し、OAuth 2.0 プロバイダーと OIDC プロバイダー両方の `userInfo` エンドポイントもチェックします。Amazon Cognito では、ID トークン内の情報は `userInfo` からの情報よりも優先されます。

プロフィールにリンクした外部ユーザーアカウントをユーザーが使用しなくなった場合は、ユーザーアカウントとユーザープールユーザーとの関連付けを解除できます。ユーザーをリンクしたときに、ユーザーの属性名、属性値、プロバイダー名をリクエストに入力しました。ユーザーに不要となったプロフィールを削除するには、同じパラメータを指定して [AdminDisableProviderForUser](#) API リクエストを行います。

AWS SDK での追加のコマンド構文と例については、「[AdminLinkProviderForUser](#)」を参照してください。

ユーザープールのマネージドログイン

ユーザープールのサービスをホストするウェブドメインを選択できます。ドメインを追加すると、Amazon Cognito ユーザープールで、以下の機能が利用可能になります。これらの機能を総称してマネージドログインと呼びます。

- [認可サーバー](#)。OAuth 2.0 および OpenID Connect (OIDC) と連携するアプリケーションに対して ID プロバイダー (IdP) として機能します。認可サーバーは、[リクエストのルーティング](#)、[JSON ウェブトークン \(JWT\) の発行と管理](#)、[ユーザー属性情報の配信](#)を行います。
- すぐに使用できるユーザーインターフェイス (UI)。サインイン、サインアウト、パスワード管理などの認証オペレーション用です。マネージドログインページは、認証サービスのウェブフロントエンドとして機能します。
- サービスプロバイダー (SP) または依頼しているパーティ (RP)。SAML 2.0 IdP、OIDC IdP、Facebook、Amazon でログイン、Apple でサインイン、Google に役立ちます。

マネージドログインと一部の機能を共有する追加のオプションとして、クラシックのホストされた UI があります。クラシックのホストされた UI は、マネージドログインサービスの第 1 世代バージョンです。ホストされた UI の IdP および RP サービスは、通常、マネージドログインと同じ特性を備えていますが、ログインページの設計がよりシンプルで、機能数も少なくなっています。例えば、パスキーサインインは、クラシックのホストされた UI では利用できません。ライト[機能プラン](#)の場合、クラシックのホストされた UI は、ユーザープールドメインサービスの唯一のオプションです。

マネージドログインページは、ユーザープールでの基本的なサインアップ、サインイン、多要素認証、およびパスワードリセットの各アクティビティを実行するためのウェブインターフェイスの集合です。また、ユーザーにサインインオプションの選択肢を提供する際に、ユーザーを 1 つ以上のサードパーティ ID プロバイダー (IdP) に接続します。アプリケーションは、ユーザーの認証と認可が必要なときに、ユーザーのブラウザでマネージドログインページを起動できます。

マネージドログインのユーザーエクスペリエンスは、カスタムロゴ、背景、スタイルを使用してブランディングに合わせることができます。マネージドログイン UI に適用できるブランディングの 2 つのオプションとして、マネージドログイン用のブランディングエディタと、ホストされた UI 用のホステッド UI (クラシック) ブランディングがあります。

ブランディングエディタ

Amazon Cognito コンソールの最新の認証オプションとビジュアルエディタを反映して更新されたユーザーエクスペリエンス。

ホステッド UI ブランディング

Amazon Cognito ユーザープールの従来の利用者には、使い慣れたユーザーエクスペリエンスです。ホステッド UI ブランディングは、ファイルベースのシステムです。ホストされた UI ページにブランディングを適用するには、ロゴイメージファイルと、いくつかの事前定義された CSS スタイルオプションの値を設定するファイルをアップロードします。

ブランディングエディタは、ユーザープールのすべての機能プランで利用できるわけではありません。詳細については、「[ユーザープールの機能プラン](#)」を参照してください。

マネージドログインおよびホストされた UI のサービスへのリクエストを構築する方法の詳細については、「[ユーザープールのエンドポイントとマネージドログインのリファレンス](#)」を参照してください。

Note

Amazon Cognito マネージドログインは、[カスタム認証チャレンジの Lambda トリガー](#)を使用したカスタム認証をサポートしていません。

トピック

- [マネージドログインのローカリゼーション](#)
- [規約ドキュメント](#)
- [を使用したマネージドログインの設定 AWS Amplify](#)
- [Amazon Cognito コンソールでのマネージドログインの設定](#)
- [サインインページの表示](#)
- [認証ページのカスタマイズ](#)

- [マネージドログインとホストされた UI について知っておくべきこと](#)
- [ユーザープールのドメインを設定する](#)
- [マネージドログインページにブランディングを適用する](#)

マネージドログインのローカリゼーション

マネージドログインでは、ユーザーインタラクティブページのデフォルト言語が英語になっています。マネージドログインページは、希望する言語にローカライズして表示できます。利用可能な言語は、AWS マネジメントコンソールで利用できる言語です。ユーザーに配信するリンクに、次の例に示すように、lang クエリパラメータを追加してください。

```
https://<your domain>/oauth2/authorize?lang=es&response_type=code&client_id=<your app client id>&redirect_uri=<your relying-party url>
```

Amazon Cognito は、lang パラメータを含む最初のリクエストを受け取ると、その言語設定を Cookie としてユーザーのブラウザに設定します。Cookie が設定されると、言語の選択は保持されるため、以降のリクエストではパラメータを表示したり、含めたりする必要がなくなります。例えば、サインインリクエストに lang=de パラメータを含めると、Cookie をクリアするか、lang=en などの新しいローカリゼーションパラメータを使用して新しいリクエストを行うまで、マネージドログインページはドイツ語で表示されます。

ローカリゼーションはマネージドログインでのみ可能です。[マネージドログインブランディング](#)を使用するには、[機能プラン](#)がエッセンシャルまたはプラスであり、ドメインを割り当てている必要があります。

ユーザーがマネージドログインで行った選択は、[カスタム E メールまたは SMS 送信者トリガー](#)では使用できません。これらのトリガーを実装するときは、他のメカニズムを使用してユーザーの希望する言語を特定する必要があります。サインインフローでは、locale 属性により、ユーザーの所在地に応じた言語が示される場合があります。サインアップフローでは、ユーザープールのリージョンまたはアプリケーションクライアント ID により、言語設定が示される場合があります。

以下の言語を使用できます。

マネージドログインの言語

Language	コード
German	de

Language	コード
英語	en
Spanish	es
French	fr
バハサインドネシア語	id
イタリア語	これ
日本語	ja
韓国語	ko
ポルトガル語 (ブラジル)	pt-BR
簡体字中国語	zh-CN
繁体字中国語	zh-TW

規約ドキュメント

ユーザーのサインアップ時に、利用規約やプライバシーポリシードキュメントへのリンクを表示するように、マネージドログインページを設定できます。アプリケーションクライアントで両方の規約ドキュメントを設定すると、ユーザーのサインアップ時に「サインアップすることで、利用規約とプライバシーポリシーに同意したものとみなされます」というテキストが表示されます。マネージドログインページに [利用規約] と [プライバシーポリシー] の文字が表示され、ドキュメントにハイパーリンクされます。

規約ドキュメントは、マネージドログインのローカリゼーションに対応した言語固有の URL をサポートしています。ユーザーが lang クエリパラメータを使用して言語を選択すると、Amazon Cognito はその言語の規約ドキュメントへのリンクを表示します。特定の言語の URL を設定していない場合、Amazon Cognito はアプリケーションクライアント用に設定したデフォルトの URL を使用します。

アプリケーションクライアントの規約ドキュメントを設定するには、ユーザープールの [マネージドログイン] メニューに移動します。[規約ドキュメント] で [規約ドキュメントを作成] を選択します。

Amazon Cognito console

規約ドキュメントを作成するには

1. ユーザープールに移動し、[マネージドログイン] メニューを選択します。[規約ドキュメント] を見つけます。
2. [規約ドキュメントを作成] を選択します。
3. 規約ドキュメントを割り当てる先のアプリケーションクライアントを選択します。
4. [規約名] に入力します。コンソールでドキュメントを識別するために使用します。
5. [リンク] の下の [言語] で言語を選択し、[URL] にその言語で規約ドキュメントをホストする URL を入力します。
6. 他の言語の URL を追加するには、[別のものを追加] を選択します。
7. [作成] を選択します。

Amazon Cognito user pools API

[CreateTerms](#) リクエスト本文の例を次に示します。これにより、マネージドログインが該当言語にローカライズされている場合、アプリケーションクライアント 1example23456789 のサインアップページには、フランス語とポルトガル語 (ブラジル) バージョンのプライバシーポリシーへのリンクが表示されます。マネージドログインのサインアップページにリンクを表示するには、事前に `terms-of-use` の URL を設定するための別のリクエストが必要です。

```
{
  "ClientId": "1example23456789",
  "Enforcement": "NONE",
  "Links": {
    "cognito:default" : "https://example.com/privacy/",
    "cognito:french" : "https://example.com/fr/privacy/",
    "cognito:portuguese-brazil" : "https://example.com/pt/privacy/"
  },
  "TermsName": "privacy-policy",
  "TermsSource": "LINK",
  "UserPoolId": "us-east-1_EXAMPLE"
}
```

 Note

Amazon Cognito のマネージドログインページに規約ドキュメントを表示するには、事前にアプリケーションクライアントの利用規約とプライバシーポリシードキュメントの両方を作成する必要があります。

を使用したマネージドログインの設定 AWS Amplify

AWS Amplify を使用してウェブまたはモバイルアプリに認証を追加する場合は、Amplify コマンドラインインターフェイス (CLI) のマネージドログインページと Amplify フレームワークのライブラリを設定できます。アプリケーションに認証を追加するには、Auth カテゴリをプロジェクトに追加します。次に、アプリケーションで Amplify クライアントライブラリを使用してユーザープールユーザーを認証します。

認証のためにマネージドログインページを呼び出すことも、IdP にリダイレクトする承認エンドポイントを介してユーザーをフェデレーションすることもできます。ユーザーがプロバイダーで正常に認証されると、Amplify は、ユーザープールに新しいユーザーを作成し、ユーザーのトークンをアプリケーションに渡します。

次の例は、AWS Amplify を使用してアプリでソーシャルプロバイダーによるマネージドログインを設定する方法を示しています。

- [React](#)
- [Swift](#)
- [Flutter](#)
- [Android](#)

Amazon Cognito コンソールでのマネージドログインの設定

マネージドログインとホストされた UI の最初の要件は、ユーザープールドメインです。ユーザープールコンソールで、ユーザープールの[ドメイン] タブに移動し、Cognito ドメインまたはカスタムドメインを追加します。新しいユーザープールを作成するプロセス中にドメインを選択することもできます。詳細については、「[ユーザープールのドメインを設定する](#)」を参照してください。ユーザープールでドメインがアクティブな場合、すべてのアプリケーションクライアントは、そのドメインでパブリック認証ページを提供します。

ユーザープールドメインを作成または変更するときに、ドメインのブランディングバージョンを設定します。ブランディングバージョンとして、[マネージドログイン] または [ホストされた UI (クラシック)] のいずれかを選択できます。選択したブランディングバージョンは、ドメインでサインインサービスを使用するすべてのアプリケーションクライアントに適用されます。

次のステップでは、ユーザープールの [\[アプリケーションクライアント\]](#) タブで [アプリケーションクライアント] を作成します。アプリケーションクライアントを作成するプロセスで、Amazon Cognito はアプリケーションに関する情報の入力と、リターン URL の選択を求めます。リターン URL は、依拠しているパーティ (RP) の URL、リダイレクト URI、コールバック URL と呼ばれます。これは、`https://www.example.com` や `myapp://example` など、アプリケーションの実行元の URL です。

ユーザープールでブランディングスタイルを使用してドメインとアプリケーションクライアントを設定すると、マネージドログインページがインターネットで利用可能になります。

サインインページの表示

Amazon Cognito コンソールの [アプリケーションクライアント] メニューで、アプリケーションクライアントの [ログインページ] タブにある [ログインページを表示] ボタンを選択します。このボタンを選択すると、ユーザープールドメインのサインインページに移動し、以下の基本パラメータが表示されます。

- アプリクライアント ID
- 認可コード付与リクエスト
- 現在のアプリクライアント用に有効にしたすべてのスコープのリクエスト
- 現在のアプリクライアントのリストにある最初のコールバック URL

[ログインページを表示] ボタンは、マネージドログインページの基本機能をテストする場合に便利です。ログインページは、[ユーザープールドメイン](#) に割り当てたブランディングバージョンと一致します。パラメータを追加または変更して、サインイン URL をカスタマイズできます。ほとんどの場合、[ログインページを表示] リンクの自動生成されたパラメータは、アプリのニーズに完全には一致しません。このような場合は、ユーザーのサインイン時にアプリが呼び出す URL をカスタマイズする必要があります。サインインパラメータのキーと値の詳細については、「[ユーザープールのエンドポイントとマネージドログインのリファレンス](#)」を参照してください。

サインインウェブページでは、次の URL 形式を使用します。この例では、`response_type=code` パラメーターを使って認可コードの付与をリクエストしています。

```
https://<your domain>/oauth2/authorize?response_type=code&client_id=<your app client id>&redirect_uri=<your relying-party url>
```

ユーザープールの [ドメイン] メニューでは、ユーザープールドメイン文字列を検索できます。[アプリケーションクライアント] メニューでは、アプリケーション ID、コールバック URL、許可されているスコープ、その他の設定を確認できます。

カスタムパラメータを使用して /oauth2/authorize エンドポイントに移動すると、Amazon Cognito は /oauth2/login エンドポイントにリダイレクトするか、identity_provider または idp_identifier パラメータがある場合は、IdP サインインページにそのままリダイレクトします。

暗黙的な付与のリクエスト例

response_type=token を使用した暗黙的なコード付与で、次の URL により、サインインウェブページを表示できます。サインインが正常に行われると、Amazon Cognito がユーザープールトークンをウェブブラウザのアドレスバーに返します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/authorize?
response_type=token&client_id=1example23456789&redirect_uri=https://
mydomain.example.com
```

ID トークンとアクセストークンは、リダイレクト URL に追加されるパラメーターとして表示されます。

ここでは、暗黙的な許可リクエストからのレスポンス例を示します。

```
https://auth.example.com/
#id_token=eyJraaBcDeF1234567890&access_token=eyJraGhIjKlM1112131415&expires_in=3600&token_type=
```

認証ページのカスタマイズ

Amazon Cognito は、これまでクラシックのホストされた UI のみを使用して、ログインページをホストしていました。ホストされた UI は、シンプルなデザインであり、認証ウェブページは一般的な外観でした。Amazon Cognito ユーザープールをカスタマイズするためにロゴイメージを使用した

り、ファイルでいくつかの CSS スタイル値を指定して一部のスタイルを微調整したりすることができました。その後、Amazon Cognito は、ホストされた認証サービスを更新して、マネージドログインを導入しました。マネージドログインは、ブランディングエディタを備え、外観と操作感が更新されています。ブランディングエディタはノーコードのビジュアルエディタであり、ホストされた UI のカスタマイズエクスペリエンスよりも多くのオプションで構成されています。マネージドログインでは、カスタム背景画像とダークモードのテーマも導入されました。

ユーザープールでは、マネージドログインとホストされた UI 間でブランディングエクスペリエンスを切り替えることができます。マネージドログインページのカスタマイズの詳細については、「[マネージドログインページにブランディングを適用する](#)」を参照してください。

マネージドログインとホストされた UI について知っておくべきこと

マネージドログインとホストされた UI の 1 時間のセッション Cookie

ユーザーがログインページまたはサードパーティプロバイダーからサインインすると、Amazon Cognito はユーザーのブラウザに Cookie を設定します。この Cookie を使用すると、1 時間以内なら、ユーザーは同じ認証方法で再度サインインできます。ブラウザ Cookie を使用してサインインすると、アプリケーションクライアント設定で指定した期間にわたり、ユーザーは新しいトークンを付与されます。ユーザー属性または認証要素を変更しても、ブラウザ Cookie を使用して再度サインインする機能には影響しません。

セッション Cookie による認証では、Cookie の期間は別の 1 時間にリセットされません。最後に成功したインタラクティブ認証から 1 時間を超えてサインインページにアクセスする場合、ユーザーは再度サインインする必要があります。

ユーザーアカウントの確認とユーザー属性の検証

ユーザープールの[ローカルユーザー](#)の場合、マネージドログインとホストされた UI は、ユーザープールを [Cognito が検証と確認のためにメッセージを自動的に送信することを許可] に設定したときに最も効果的です。この設定を有効にすると、Amazon Cognito は、サインアップしたユーザーに確認コードを含むメッセージを送信します。代わりにユーザーをユーザープール管理者として確認すると、サインアップ後にログインページにエラーメッセージが表示されます。この状態で、Amazon Cognito は新しいユーザーを作成しましたが、検証メッセージを送信できませんでした。引き続きユーザーを管理者として確認することはできますが、エラーが発生すると、ユーザーはサポートデスクに連絡する可能性があります。管理者の確認の詳細については、「[ユーザーにアプリケーションへのサインアップを許可するがユーザープール管理者として確認する](#)」を参照してください。

マネージドログインのオペレーションの範囲

マネージドログインとクラシックのホストされた UI は、サインアップ、サインイン、パスワード管理をサポートしています。これには、多要素認証 (MFA) によるサインインの完了と WebAuthn 認証子の登録が含まれます。マネージドログインは、属性の変更や MFA の設定構成など、ユーザーによるセルフサービスのプロファイル管理をサポートしていません。プロファイル管理は、独自のアプリケーションコードに実装する必要があります。また、マネージドログインでは、[AdminUpdateUserAttributes](#) API オペレーションを使用して管理者として E メールアドレスと電話番号を更新するときに、属性の変更を確認する機能も提供されていません。

設定の変更の表示

ページのスタイルを変更しても、変更がすぐに反映されない場合は、数分間待ってから、ページを更新してください。

ユーザープールトークンのデコード

Amazon Cognito ユーザープールトークンは、RS256 アルゴリズムを使用して署名されます。を使用して、ユーザープールトークンをデコードおよび検証できます AWS Lambda。GitHub の「[Amazon Cognito JWT トークンのデコードと検証](#)」を参照してください。

TLS バージョン

マネージドログインページとホストされた UI ページには、転送中の暗号化が必要です。Amazon Cognito が提供するユーザープールドメインでは、ユーザーのブラウザが TLS バージョン 1.2 以上をネゴシエートする必要があります。カスタムドメインは、TLS バージョン 1.2 でのブラウザ接続をサポートしています。ホストされた UI (クラシック) では、カスタムドメインに TLS 1.2 は不要ですが、新しいマネージドログインでは、カスタムドメインとプレフィックスドメインの両方に TLS バージョン 1.2 が必要です。ドメインサービスの設定は、Amazon Cognito が管理するため、ユーザープールドメインの TLS 要件を変更することはできません。

CORS ポリシー

マネージドログインとホストされた UI は、どちらも、カスタムのクロスオリジンリソース共有 (CORS) オリジンポリシーをサポートしていません。CORS ポリシーにより、ユーザーがリクエストで認証パラメータを渡すことは許可されません。代わりに、アプリケーションのフロントエンドに CORS ポリシーを実装してください。Amazon Cognito は、以下のエンドポイントへのリクエストに対して Access-Control-Allow-Origin: * レスponseヘッダーを返します。

1. [トークンエンドポイント](#)
2. [エンドポイントの取り消し](#)
3. [userInfo エンドポイント](#)

cookie

マネージドログインとホストされた UI は、ユーザーのブラウザに Cookie を設定します。Cookie は、サイトがサードパーティーの Cookie を設定しない一部のブラウザの要件に準拠しています。これらはユーザープールエンドポイントのみを対象とし、以下が含まれます。

- 各リクエストの XSRF-TOKEN Cookie。
- ユーザーがリダイレクトされたときのセッション整合性のための csrf-state Cookie。
- セッション整合性のための csrf-state-legacy Cookie。ブラウザが SameSite 属性をサポートしていない場合に Amazon Cognito がフォールバックとして読み取ります。
- 成功したサインイン試行を 1 時間保持する cognito セッション Cookie。
- マネージドログインでユーザーが選択した[言語ローカリゼーション](#)を保持する lang Cookie。
- ユーザーがマネージドログインページ間を移動するときに必須データを保持する page-data Cookie。

iOS では、[すべての Cookie をブロック](#)できます。この設定は、マネージドログインまたはホストされた UI とは互換性がありません。この設定を有効にする可能性のあるユーザーと連携するには、AWS SDK を使用してユーザープール認証をネイティブ iOS アプリに構築します。このシナリオでは、Cookie ベースではない独自のセッションストレージを構築できます。

マネージドログインのバージョン変更の影響

ドメインを追加し、マネージドログインのバージョンを設定する場合、以下の影響を考慮してください。

- マネージドログインまたはホストされた UI (クラシック) のいずれかのブランディングを使用してプレフィックスドメインを追加した場合、ログインページが使用可能になるまでに最大 60 秒かかることがあります。
- マネージドログインまたはホストされた UI (クラシック) のブランディングを使用してカスタムドメインを追加した場合、ログインページが使用可能になるまでに最大 5 分かかることがあります。
- ドメインのブランディングバージョンを変更した場合、ログインページが新しいブランディングバージョンで利用可能になるまでに最大 4 分かかることがあります。
- マネージドログインとホストされた UI (クラシック) のブランディングを切り替えると、Amazon Cognito はユーザーセッションを維持しません。ユーザーは、新しいインターフェイスで再度サインインする必要があります。

デフォルトのスタイル

でアプリケーションを作成すると AWS マネジメントコンソール、Amazon Cognito はアプリケーションにブランドスタイルを自動的に割り当てます。[CreateUserPoolClient](#) オペレーションを使用してプログラムでアプリケーションクライアントを作成した場合、ブランディングスタイルは作成されません。マネージドログインは、[CreateManagedLoginBranding](#) リクエストでアプリケーションクライアントを作成するまで、AWS SDK で作成されたアプリケーションクライアントでは使用できません。

マネージドログイン認証プロンプトのタイムアウト

Amazon Cognito は認証リクエストが 5 分以内に完了しない場合にこれをキャンセルし、ユーザーをマネージドログインにリダイレクトします。ページには、Something went wrong というエラーメッセージが表示されます。

ユーザープールのドメインを設定する

ドメインの設定は、ユーザープールを設定するオプションです。ユーザープールドメインは、ユーザー認証、サードパーティプロバイダーとのフェデレーション、および OpenID Connect (OIDC) フローについての機能をホストします。マネージドログインは、サインアップ、サインイン、パスワード復旧など、主要なオペレーション用の事前構築済みのインターフェイスです。また、Machine to Machine (M2M) 認可、その他の OIDC や OAuth 2.0 の認証と認可フローのために、[承認](#)、[ユーザー情報](#)、[トークン](#)など、OpenID Connect (OIDC) の標準エンドポイントもホストします。

ユーザーは、ユーザープールに関連付けられたドメインでマネージドログインページを使用して認証します。このドメインを設定するには、次の 2 つのオプションがあります。デフォルトの Amazon Cognito ホストドメインを使用することも、所有するカスタムドメインを設定することもできます。

カスタムドメインオプションには、柔軟性、セキュリティ、制御のためのオプションが多数用意されています。例えば、使い慣れた組織所有のドメインは、ユーザーの信頼を高めて、サインインプロセスが直感的にできます。ただし、カスタムドメインアプローチには、SSL 証明書の管理や DNS 設定など、追加のオーバーヘッドが必要です。

OIDC 検出エンドポイント (エンドポイント URL 用の /.well-known/openid-configuration、およびトークン署名キー用の /.well-known/jwks.json) はドメインでホストされません。詳細については、「[ID プロバイダーと依拠しているパーティーエンドポイント](#)」を参照してください。

ユーザープールのドメインを設定および管理する方法を理解することは、認証をアプリケーションに統合するための重要なステップです。ユーザープール API と AWS SDK を使用してサインインす

ることは、ドメインを設定する代わりに使用できます。API ベースのモデルは API レスポンスに直接トークンを配信しますが、ユーザープールの拡張機能を OIDC IdP として使用する実装の場合は、ドメインを設定する必要があります。ユーザープールで利用できる認証モデルの詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

トピック

- [ユーザープールドメインについて知っておくべきこと](#)
- [マネージドログインに Amazon Cognito プレフィックスドメインを使用する](#)
- [マネージドログインに独自のドメインを使用する](#)

ユーザープールドメインについて知っておくべきこと

ユーザープールドメインは、アプリケーション内の OIDC の依拠しているパーティー向け、および UI 向けのサービスポイントです。ユーザープールのドメインの実装を計画する場合は、次の詳細を考慮してください。

予約済みの用語

Amazon Cognito プレフィックスドメイン名で aws、amazon、または cognito というテキストを使用することはできません。

検出エンドポイントが別のドメインにある

ユーザープール[検出エンドポイント](#)の .well-known/openid-configuration および .well-known/jwks.json は、ユーザープールのカスタムドメインまたはプレフィックスドメインにはありません。これらのエンドポイントへのパスは次のとおりです。

- `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`

ドメイン変更の所要時間

Amazon Cognito がプレフィックスドメインのブランディングバージョンを起動または更新するまでに最大 1 分かかる場合があります。カスタムドメインへの変更は、反映されるまでに最大 5 分かかる場合があります。新しいカスタムドメインは、反映されるまで最大 1 時間かかる場合があります。

カスタムドメインとプレフィックスドメインを同時に

カスタムドメインとが所有するプレフィックスドメインの両方を使用してユーザープールを設定できます AWS。ユーザープール[検出エンドポイント](#)は別のドメインでホストされるため、カスタムドメインにのみ対応します。例えば、openid-configuration は "https://auth.example.com/oauth2/authorize" の "authorization_endpoint" に 1 つの値を提供します。

ユーザープールにカスタムドメインとプレフィックスドメインの両方がある場合、OIDC プロバイダーのすべての機能でカスタムドメインを使用できます。この設定を持つユーザープールのプレフィックスドメインには検出エンドポイントやトークン署名キーエンドポイントがないため、それに応じて使用する必要があります。

パスキーの依拠しているパーティの ID として優先されるカスタムドメイン

[パスキー](#)でユーザープール認証を設定するときは、依拠しているパーティ (RP) の ID を設定する必要があります。カスタムドメインとプレフィックスドメインがある場合、RP ID はカスタムドメインとしてのみ設定できます。Amazon Cognito コンソールでプレフィックスドメインを RP ID として設定するには、カスタムドメインを削除するか、プレフィックスドメインの完全修飾ドメイン名 (FQDN) をサードパーティのドメインとして入力します。

ドメイン階層の異なるレベルでカスタムドメインを使用しない

auth.example.com や auth2.example.com など、同じ最上位ドメイン (TLD) にカスタムドメインを持つように個別のユーザープールを設定できます。マネージドログインのセッション Cookie は、カスタムドメインと、*.auth.example.com などのすべてのサブドメインに対して有効です。このため、アプリケーションのユーザーは、親ドメインおよびサブドメインでマネージドログインにアクセスできません。カスタムドメインが同じ TLD を使用している場合は、同じサブドメインレベルで保持します。

カスタムドメイン auth.example.com を持つユーザープールがあるとします。次に、別のユーザープールを作成し、カスタムドメイン uk.auth.example.com を割り当てるとします。ユーザーは auth.example.com でサインインし、ワイルドカードパス *.auth.example.com でブラウザが任意のウェブサイトに提示する Cookie を取得します。その後、uk.auth.example.com へのサインインを試みます。ユーザープールドメインに無効な Cookie が渡され、サインインプロンプトの代わりにエラーが返されます。対照的に、*.auth.example.com 用の Cookie を持つユーザーは、auth2.example.com でサインインセッションを開始しても問題ありません。

ブランディングバージョン

ドメインを作成するときは、ブランディングバージョンを設定します。オプションは、新しいマネージドログインエクスペリエンスとクラシックのホストされた UI エクスペリエンスです。この選択は、ドメインでサービスをホストするすべてのアプリケーションクライアントに適用されます。

マネージドログインに Amazon Cognito プレフィックスドメインを使用する

マネージドログインのデフォルトのエクスペリエンスは、AWS が所有するドメインでホストされます。このアプローチでは、プレフィックス名を選択してアクティブにできるので、エントリに対する障壁は低くなりますが、カスタムドメインの信頼感を高める機能はありません。Amazon Cognito ドメインオプションとカスタムドメインオプションの間でコスト差はありません。唯一の違いは、ユーザーに指示するウェブアドレスのドメインです。サードパーティーの IdP リダイレクトとクライアント認証情報フローの場合、ホストされたドメインはほとんど表示されません。ユーザーがマネージドログインを使用してサインインし、アプリケーションドメインと一致しない認証ドメインとやり取りする場合は、カスタムドメインの方が適しています。

ホストされた Amazon Cognito ドメインには、選択したプレフィックスがありますが、ルートドメイン `amazoncognito.com` でホストされます。以下に例を示します。

```
https://cognitoexample.auth.ap-south-1.amazoncognito.com
```

すべてのプレフィックスドメインは、次の形式に従います: `prefix.auth.AWS #####code.amazoncognito.com`。[カスタムドメイン](#) ユーザープールは、マネージドログインまたはホストされた UI ページを、所有している任意のドメインでホストできます。

Note

Amazon Cognito アプリケーションのセキュリティを強化するために、ユーザープールエンドポイントの親ドメインは[パブリックサフィックスリスト \(PSL\)](#)に登録されます。PSL は、ユーザーのウェブブラウザが、ユーザーが設定するユーザープールエンドポイントと Cookie を一貫して把握するのに役立ちます。

ユーザープールの親ドメインは、以下の形式になります。

```
auth.Region.amazoncognito.com  
auth-fips.Region.amazoncognito.com
```

でアプリケーションクライアントとユーザープールドメインを追加するには AWS マネジメントコンソール、「」を参照してください[アプリクライアントの作成](#)。

トピック

- [前提条件](#)
- [Amazon Cognito ドメインプレフィックスを設定する](#)
- [サインインページを検証する](#)

前提条件

開始するには、以下が必要です。

- アプリクライアントを持つユーザープール。詳細については、「[ユーザープールの開始方法](#)」を参照してください。

Amazon Cognito ドメインプレフィックスを設定する

ユーザープールドメインを設定するには、AWS マネジメントコンソール または AWS CLI または API を使用できます。

Amazon Cognito console

ドメインを設定する

1. [ブランディング] の下にある [ドメイン] メニューに移動します。
2. [ドメイン] の横で [アクション]、[Cognito ドメインを作成] の順に選択します。ユーザープールのプレフィックスドメインを設定済みである場合は、新しいカスタムドメインを作成する前に、[Cognitoドメインの削除] を選択します。
3. Amazon Cognito ドメインで使用するための使用可能なドメインプレフィックスを入力します。カスタムドメインの設定方法については、「[マネージドログインに独自のドメインを使用する](#)」を参照してください。
4. [ブランディングバージョン] を選択します。ブランディングバージョンは、そのドメインのすべてのユーザーインタラクティブページに適用されます。ユーザープールは、すべてのアプリケーションクライアントでマネージドログインブランディングまたはホステッド UI ブランディングのいずれかをホストできます。

 Note

カスタムドメインとプレフィックスドメインを持つことができますが、Amazon Cognito はカスタムドメイン用の /.well-known/openid-configuration エンドポイントのみを提供します。

5. [作成] を選択します。

CLI/API

ドメインプレフィックスを作成してユーザープールに割り当てるには、次のコマンドを使用します。

ユーザープールのドメインを設定する

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

例: `aws cognito-idp create-user-pool-domain --user-pool-id
<user_pool_id> --domain <domain_name> --managed-login-version 2`

- ユーザープールの API オペレーション: [CreateUserPoolDomain](#)

ドメインに関する情報を取得する

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

例: `aws cognito-idp describe-user-pool-domain --domain <domain_name>`

- ユーザープールの API オペレーション: [DescribeUserPoolDomain](#)

ドメインを削除する

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

例: `aws cognito-idp delete-user-pool-domain --domain <domain_name>`

- ユーザープールの API オペレーション: [DeleteUserPoolDomain](#)

サインインページを検証する

- Amazon Cognito でホストされるドメインから、サインインページが利用可能であることを検証します。

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

ドメインは、Amazon Cognito コンソールの [ドメイン名] ページに表示されます。アプリケーション ID およびコールバック URL は [アプリケーションの設定] ページに表示されています。

マネージドログインに独自のドメインを使用する

アプリケーションクライアントをセットアップしたら、[マネージドログイン](#)のドメインサービスでカスタムドメインのユーザープールを設定できます。カスタムドメインを使用すると、ユーザーはデフォルトの[プレフィックスドメイン](#) amazoncognito.com の代わりに、独自のウェブアドレスを使用してアプリケーションにサインインできます。カスタムドメインは、特にルートドメインがアプリケーションをホストするドメインと一致する場合、よく知られているドメイン名を使用してアプリケーションのユーザー信頼を向上させます。カスタムドメインは、組織のセキュリティ要件へのコンプライアンスを強化できます。

カスタムドメインには、ユーザープール、アプリケーションクライアント、所有するウェブドメインなど、いくつかの前提条件があります。カスタムドメインには、米国東部 (バージニア北部) の AWS Certificate Manager (ACM) で管理されるカスタムドメインの SSL 証明書も必要です。Amazon Cognito は、ACM 証明書によって転送中に保護される Amazon CloudFront ディストリビューションを作成します。ドメインを所有している場合、カスタムドメインの CloudFront ディストリビューションにトラフィックを誘導する DNS レコードを作成する必要があります。

これらの要素の準備ができれば、Amazon Cognito のコンソールまたは API を使用してカスタムドメインをユーザープールに追加できます。これには、ドメイン名と SSL 証明書を指定したうえで、指定したエイリアスターゲットで DNS 設定を更新する必要があります。これらの変更を行った後、サインインページがカスタムドメインでアクセス可能であることを検証できます。

カスタムドメインを作成する最も簡単な方法は、Amazon Route 53 のパブリックホストゾーンを使用することです。Amazon Cognito コンソールでは、適切な DNS レコードを数ステップで作成できます。開始する前に、所有しているドメインまたはサブドメイン用の [Route 53 ホストゾーンを作成する](#)ことを検討してください。

トピック

- [ユーザープールへのカスタムドメインの追加](#)
- [前提条件](#)
- [ステップ 1: カスタムドメイン名を入力する](#)
- [ステップ 2: エイリアスターゲットとサブドメインを追加する](#)
- [ステップ 3: サインインページを検証する](#)
- [カスタムドメインの SSL 証明書を変更する](#)

ユーザープールへのカスタムドメインの追加

カスタムドメインをユーザープールに追加するには、Amazon Cognito コンソールでドメイン名を指定し、[AWS Certificate Manager](#) (ACM) を使用してユーザー管理の証明書を提供します。ドメインを追加したら、Amazon Cognito がエイリアスターゲットを提供するので、これを DNS 設定に追加します。

前提条件

開始するには、以下が必要です。

- アプリクライアントを持つユーザープール。詳細については、「[ユーザープールの開始方法](#)」を参照してください。
- 所有するウェブドメイン。その親ドメインには、DNS に有効な A レコードが必要です。このレコードには任意の値を割り当てることができます。親は、ドメインのルート、またはドメイン階層の 1 つ上の子ドメインです。例えば、カスタムドメインが `auth.xyz.example.com` である場合、Amazon Cognito は `xyz.example.com` を IP アドレスに解決できることが必要です。お客様のインフラストラクチャへの偶発的な影響を防ぐため、Amazon Cognito は、最上位ドメイン (TLD) のカスタムドメインへの使用をサポートしていません。詳細については、「[ドメイン名](#)」を参照してください。
- カスタムドメインのサブドメインを作成する機能。サブドメイン名としては、`auth` をお勧めします。たとえば、「`auth.example.com`」を使用します。

Note

「[ワイルドカード証明書](#)」がない場合は、カスタムドメインのサブドメイン用に新しい証明書の取得が必要な場合があります。

- 米国東部 (バージニア北部) の ACM によって管理されるパブリック SSL/TLS 証明書。証明書はグローバルサービスである CloudFront のディストリビューションに関連付けられるため、証明書は us-east-1 に配置する必要があります。
- Server Name Indication (SNI) をサポートするブラウザクライアント。Amazon Cognito がカスタムドメインに割り当てる CloudFront ディストリビューションには SNI が必要です。この設定は変更できません。CloudFront ディストリビューションの SNI の詳細については、「Amazon CloudFront デベロッパーガイド」の「[SNI を使用して HTTPS リクエストを処理する](#)」を参照してください。
- ユーザープール認可サーバーがユーザーセッションに Cookie を追加することを許可するアプリケーション。Amazon Cognito は、マネージドログインページに必要ないくつかの Cookie を設定します。cognito、cognito-fl、XSRF-TOKEN などです。個々の Cookie はブラウザのサイズ制限に適合していますが、ユーザープールの設定を変更すると、マネージドログインの Cookie のサイズが大きくなる可能性があります。カスタムドメインの前の Application Load Balancer (ALB) などの中間サービスでは、ヘッダーの最大サイズまたは Cookie の合計サイズが適用される場合があります。アプリケーションが独自の Cookie も設定している場合、ユーザーのセッションはこれらの制限を超える可能性があります。サイズ制限の競合を避けるため、アプリケーションでは、ユーザープールのドメインサービスをホストするサブドメインに Cookie を設定しないようお勧めします。
- Amazon CloudFront ディストリビューションを更新するアクセス許可。そのためには、次の IAM ポリシーステートメントを AWS アカウントのユーザーにアタッチします。

JSON

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

CloudFront でのアクションの承認の詳細については、「[CloudFront でのアイデンティティベースのポリシー \(IAM ポリシー\) の使用](#)」を参照してください。

Amazon Cognito は、最初に IAM アクセス許可を使用して CloudFront デイストリビューションを設定しますが、デイストリビューションは AWS によって管理されます。Amazon Cognito がユーザープールに関連付けた CloudFront デイストリビューションの設定を変更することはできません。例えば、セキュリティポリシーでサポートされている TLS バージョンを更新することはできません。

ステップ 1: カスタムドメイン名を入力する

ドメインは、Amazon Cognito コンソールまたは API を使用することでユーザープールに追加できます。

Amazon Cognito console

Amazon Cognito コンソールからユーザープールにドメインを追加する。

1. [ブランディング] の下にある [ドメイン] メニューに移動します。
2. [ドメイン] の横で、[アクション] を選択し、[カスタムドメインの作成] または [Amazon Cognito ドメインの作成] のどちらかを選択します。ユーザープールのカスタムドメインを設定済みである場合は、新しいカスタムドメインを作成する前に、[カスタムドメインを削除] を選択します。
3. [ドメイン] の横にある [アクション] を選択し、[カスタムドメインを作成] を選択します。カスタムドメインを設定済みである場合は、新しいカスタムドメインを作成する前に、[カスタムドメインを削除] を選択して既存のドメインを削除します。
4. [Custom domain] (カスタムドメイン) には、Amazon Cognito で使用するドメインの URL を入力します。ドメイン名で利用できるのは、小文字の英文字、数字、およびハイフンのみです。最初または最後の文字にハイフンを使用しないでください。サブドメイン名は、ピリオドで区切ります。
5. [ACM certificate] (ACM 証明書) には、ドメインに使用する SSL 証明書を選択します。ユーザープールの に関係なく、Amazon Cognito カスタムドメインで利用できるのは、米国東部 (バージニア北部) AWS リージョン の ACM 証明書のみです。

利用可能な証明書がない場合は、ACM を使用して米国東部 (バージニア北部) にプロビジョニングできます。詳細については、AWS Certificate Manager ユーザーガイドの「[使用開始](#)」を参照してください。

6. [ブランディングバージョン] を選択します。ブランディングバージョンは、そのドメインのすべてのユーザーインタラクティブページに適用されます。ユーザープールは、すべてのアプリケーションクライアントでマネージドログインブランディングまたはホステッド UI ブランディングのいずれかをホストできます。

 Note

カスタムドメインとプレフィックスドメインを持つことができますが、Amazon Cognito はカスタムドメイン用の /.well-known/openid-configuration エンドポイントのみを提供します。

7. [作成] を選択します。
8. Amazon Cognito は [ドメイン] メニューに戻ります。メッセージ「Create an alias record in your domain's DNS」(ドメインの DNS にエイリアスレコードを作成する) が表示されます。コンソールに表示される [Domain] (ドメイン) と [Alias target] (エイリアスターゲット) を書き留めます。これらは、次のステップでカスタムドメインにトラフィックを転送するために使用します。

API

次の [CreateUserPoolDomain](#) リクエスト本文では、カスタムドメインを作成します。

```
{
  "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-EXAMPLE11111"
  }
}
```

ステップ 2: エイリアスターゲットとサブドメインを追加する

このステップでは、ドメイン名サーバー (DNS) サービスプロバイダーを使用して、前のステップのエイリアスターゲットを指すエイリアスを設定します。DNS アドレスの解決に Amazon Route 53 を使用している場合は、Route 53 を使用してエイリアスターゲットとサブドメインを追加するセクションを選択します。

エイリアスターゲットおよびサブドメインを現在の DNS 設定に追加する

- DNS アドレス解決に Route 53 を使用していない場合は、DNS サービスプロバイダーの設定ツールを使用して、前のステップで設定したエイリアスターゲットをドメインの DNS レコードに追加する必要があります。また、DNS プロバイダーで、カスタムドメインのサブドメインを設定する必要があります。

Route 53 を使用してエイリアスターゲットとサブドメインを追加する

1. [Route 53 コンソール](#)にサインインします。プロンプトが表示されたら、AWS 認証情報を入力します。
2. Route 53 にパブリックホストゾーンがない場合は、カスタムドメインの親であるルートでパブリックホストゾーンを作成します。詳細については、「Amazon Route 53 デベロッパーガイド」の「[Creating a public hosted zone](#)」を参照してください。
 - a. [ホストゾーンの作成] を選択します。
 - b. ドメイン名のリストから、カスタムドメイン *myapp.auth.example.com* の親ドメイン *auth.example.com* を入力します。
 - c. ホストゾーンの説明を入力します。
 - d. パブリックホストゾーンのホストゾーン [Type] (タイプ) を選択して、パブリッククライアントがカスタムドメインを解決できるようにします。Private hosted zone (プライベートホストゾーン) はサポートされていません。
 - e. 必要に応じてタグを適用します。
 - f. [ホストゾーンの作成] を選択します。

Note

また、サブドメインのホストゾーンにクエリを送信する親ホストゾーン内の委任セットを持つ、カスタムドメインの新しいホストゾーンを作成することもできます。それ以外の場合は、A レコードを作成します。この方法により、ホストゾーンの柔軟性とセキュリティが向上します。詳細については、「[Amazon Route 53 を通してホストされるドメインのサブドメインの作成](#)」をご覧ください。

3. [ホストゾーン] ページで、ホストゾーンの名前を選択します。
4. カスタムドメインの親ドメインに DNS レコードがまだない場合は、DNS レコードを追加します。次のプロパティを使用して、親ドメインの DNS レコードを作成します。

- レコード名: 空白のままにします。
 - レコードタイプ: A。
 - エイリアス: 有効にしないでください。
 - 値: 選択したターゲットを入力します。このレコードは何らかの値に解決される必要がありますが、レコードの値は Amazon Cognito にとって重要ではありません。
 - TTL: 任意の TTL に設定するか、デフォルトのままにします。
 - ルーティングポリシー: [シンプルルーティング] を選択します。
5. [レコードを作成] を選択します。以下に、ドメイン *example.com* のレコードの例を示します。

example.com. 60 IN A *198.51.100.1*

 Note

Amazon Cognito は、本番ドメインの予想外の乗っ取りから保護するために、カスタムドメインの親ドメインで使用する DNS レコードがあることを確認します。親ドメインの DNS レコードがない場合、カスタムドメインを設定しようとすると、Amazon Cognito はエラーを返します。Start of Authority (SOA) レコードは、親ドメインの検証のために十分な DNS レコードではありません。

6. 次のプロパティを使用して、カスタムドメインに別の DNS レコードを追加します。
- レコード名: カスタムドメインプレフィックス。例えば、*auth.example.com* のレコードを作成するための *auth*。
 - レコードタイプ: A。
 - エイリアス: 有効にします。
 - トラフィックのルーティング先: [Cloudfront ディストリビューションへのエイリアス] を選択します。前に記録したエイリアスターゲット (例えば、*123example.cloudfront.net*) を入力します。
 - ルーティングポリシー: [シンプルルーティング] を選択します。
7. [レコードを作成] を選択します。

Note

新しいレコードがすべての Route 53 の DNS サーバーに伝播されるまでに約 60 秒かかる場合があります。Route 53 の [GetChange](#) API メソッドを使用して、変更が反映されていることが確認できます。

ステップ 3: サインインページを検証する

- カスタムドメインでサインインページが表示できることを確認します。

このアドレスをブラウザに入力して、カスタムドメインとサブドメインでサインインします。以下は、サブドメイン *auth* を使用したカスタムドメイン *example.com* のサンプル URL です。

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

カスタムドメインの SSL 証明書を変更する

必要な場合は、Amazon Cognito を使用して、カスタムドメインに適用した証明書を変更することができます。

通常、これは ACM での定期的な証明書更新後には不要です。ACM で既存の証明書を更新するときは、証明書の ARN がそのまま維持され、カスタムドメインは自動的に新しい証明書を使用します。

ただし、既存の証明書を新しい証明書に置き換える場合は、ACM が新しい証明書に新しい ARN を提供します。新しい証明書をカスタムドメインに適用するには、この ARN を Amazon Cognito に提供する必要があります。

新しい証明書の提供後、Amazon Cognito では、それがカスタムドメインに配布されるまで最大 1 時間かかります。

開始する前に

Amazon Cognito で証明書を変更する前に、証明書を ACM に追加する必要があります。詳細については、AWS Certificate Manager ユーザーガイドの「[使用開始](#)」を参照してください。

証明書を ACM に追加するときは、AWS リージョンとして米国東部 (バージニア北部) を選択する必要があります。

証明書は、Amazon Cognito コンソールまたは API を使用して変更できます。

AWS マネジメントコンソール

Amazon Cognito コンソールで証明書を更新する

1. にサインイン AWS マネジメントコンソール し、 で Amazon Cognito コンソールを開きます <https://console.aws.amazon.com/cognito/home>。
2. [User Pools] (ユーザープール) を選択します。
3. 証明書を更新するユーザープールを選択します。
4. [ドメイン] メニューを選択します。
5. [Actions] (アクション)、[Edit ACM certificate] (ACM 証明書の編集) を選択します。
6. カスタムドメインに関連付ける新しい証明書を選択します。
7. [Save changes] (変更の保存) をクリックします。

API

証明書を更新する (Amazon Cognito API)

- [UpdateUserPoolDomain](#) アクションを使用します。

マネージドログインページにブランディングを適用する

認証サービスとアプリケーションの間で一貫したユーザーエクスペリエンスを提供できます。この目標は、AWS SDK のカスタムフォームとバックエンド API オペレーション、またはマネージドログインを使用して達成できます。マネージドログインとクラシックのホストされた UI は、ユーザープールによる認証を提供するアプリケーションコンポーネントのウェブフロントエンドです。マネージド認証サービスをアプリケーション UX と同期するには、2 つのカスタマイズオプションとして、ブランディングエディタとホステッド UI ブランディングがあります。Amazon Cognito コンソールとユーザープール API オペレーションで、好みのエクスペリエンスを選択できます。

ブランディングエディタ

[ブランディングエディタ](#)は、最新のユーザープール UI エクスペリエンスである[マネージドログイン](#)用の最新のカスタマイズオプションです。ブランディングエディタは、マネージドログインのアセットおよびスタイル用のノーコードビジュアルエディタであり、一連の API オペレーションを使用して多くの設定オプションをプログラムで設定できます。ユーザープールを[ドメイン](#)とマネージドログインを使用して設定すると、ログインページのブランディングデザイナーバージョンが自動的にレンダリングされます。

ホステッド UI (クラシック) ブランディング

[ホステッド UI \(クラシック\) ブランディングエクスペリエンス](#)には、固定されたスタイルオプションセットを使用してカスタマイズスタイルシート (CSS) ファイルを変更するオプションと、カスタムロゴイメージを追加するオプションの 2 つがあります。これらのオプションは、Amazon Cognito コンソールまたは [SetUICustomization](#) API オペレーションで設定できます。サービスを開始した時点では、Amazon Cognito にはこのオプションしかありませんでした。ユーザープールを[ドメイン](#)とホステッド UI ブランディングバージョンで設定すると、ログインページのクラシックバージョンが自動的にレンダリングされます。[機能プラン](#)によっては、ホストされた UI のみをサポートする場合があります。

Note

ブランディングエディタとクラシックブランディングエクスペリエンスは、ホストされた認証サービスのビジュアルプロパティを変更します。現在、複数の言語のいずれかにローカライズする場合を除き、マネージドログインページに表示されるテキストを変更することはできません。ローカライズの詳細については、「[マネージドログインのローカリゼーション](#)」を参照してください。

ブランディングエクスペリエンスを選択してスタイルを割り当てる

Amazon Cognito コンソールでは、新しいユーザープールにデフォルトでマネージドログインのブランディングエクスペリエンスが適用されます。マネージドログインが利用可能になる前に設定したユーザープールには、ホステッド UI (クラシック) ブランディングが適用されます。マネージドログインブランディングとホステッド UI ブランディングは切り替えることができます。ブランディングバージョンを変更すると、Amazon Cognito は、その変更をユーザープールドメインのユーザーインタラクティブページに直ちに反映します。マネージドログインとホステッド UI を使用すると、ユーザープールはアプリケーションクライアントごとに独自のスタイルを持つことができます。

アプリケーションクライアントごとに独自のブランディングスタイルを持つことができますが、ユーザープールドメインは、マネージドログインまたはホステッド UI のいずれかとして機能します。スタイルは、アプリケーションクライアントに適用される一連のカスタマイズ設定です。ユーザープールごとに 1 つの [カスタムドメイン](#) と 1 つの [プレフィックスドメイン](#) を設定できます。カスタムドメインとプレフィックスドメインに異なるブランディングバージョンを割り当てることができます。ただし、カスタムドメインを同時に設定している場合、プレフィックスドメインは完全には機能しません。.well-known OIDC 検出エンドポイントは、カスタムドメインパスのみを提示します。プレフィックスドメインは、この設定のユーザープールでは、エンドポイント検出 (openid-configuration) を必要としないオペレーションにのみ使用できます。ユーザープールのこれらの特性により、ユーザープールごとに 1 つのブランディングバージョンを効果的に選択できます。

マネージドログインのブランディングバージョンに設定されているドメインがユーザープールにあると、ユーザープール内のアプリケーションクライアントにスタイルを割り当てることができます。スタイルは、イメージファイル、表示オプション、CSS 値で構成される一連のビジュアル設定です。アプリケーションクライアントにスタイルを割り当てると、Amazon Cognito はすぐに更新内容をユーザーインタラクティブログインページにプッシュします。選択したブランディングバージョンとこれに適用したカスタマイズが、ユーザーインタラクティブページに反映されます。

スタイルの更新と削除

スタイルを作成したら、アプリケーションクライアントにスタイルをリンクします。アプリケーションクライアントのスタイル割り当てを変更するには、まず元のスタイルを削除する必要があります。現在、スタイル間で設定をコピーすることはできません。この操作は、プログラムで行う必要があります。スタイルとアプリケーションクライアント間で設定をレプリケートするには、[DescribeManagedLoginBranding](#) API オペレーションを使用してスタイルの設定を取得し、[CreateManagedLoginBranding](#) または [UpdateManagedLoginBranding](#) を使用して適用します。アプリケーションクライアントの割り当て済みスタイルを変更することはできません。元のスタイルを削除して新しいスタイルを設定することのみができます。API および SDK オペレーションを使用したスタイルの管理の詳細については、「[マネージドログインのブランディングの API および SDK オペレーション](#)」を参照してください。

Note

ブランディングスタイルを作成または更新するプログラムリクエストは、リクエストサイズが 2 MB 以下である必要があります。リクエストがこの制限より大きい場合は、リクエストを複数の UpdateManagedLoginBranding リクエストに分割し、最大リクエストサイズを超えないパラメータのグループにします。これらのリクエストでは、指定していないパラ

メータがデフォルトに設定されることはないため、既存の設定に影響を与えずに、部分的なリクエストを送信できます。

スタイルを削除するには、Amazon Cognito コンソールの [マネージドログイン] メニューを使用します。[スタイル] で削除するスタイルを選択し、[スタイルを削除] を選択します。

ドメインにブランディングを割り当てるプロセスは、以下のステップに要約されます。

1. [ドメインを作成し、ブランディングバージョンを設定します。](#)
2. ブランディングスタイルを作成してアプリケーションクライアントに割り当てます。

アプリケーションクライアントにスタイルを割り当てるには

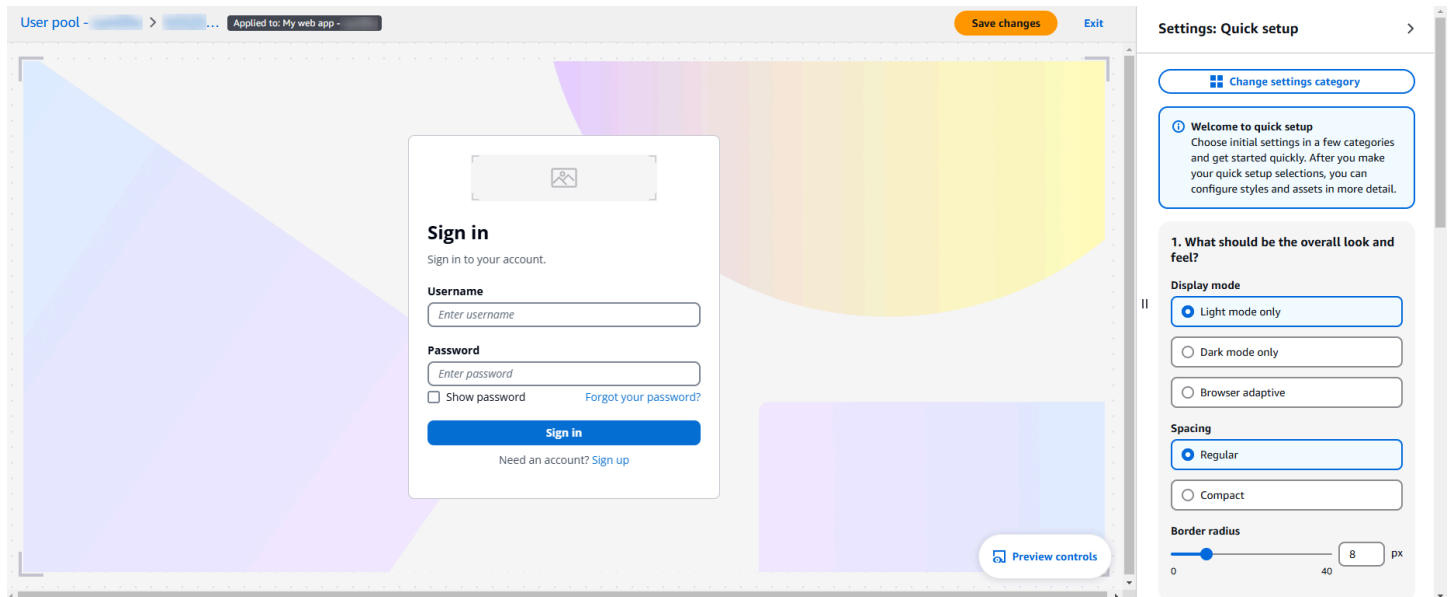
1. ユーザープールの [ドメイン] メニューでドメインを作成し、[ブランディングバージョン] を [マネージドログイン] に設定します。
2. [マネージドログイン] メニューに移動します。[スタイル] で、[スタイルを作成] を選択します。
3. スタイルを割り当てる先のアプリケーションクライアントを選択するか、新しい [アプリケーションクライアント](#) を作成します。
4. ブランディング設定の構成を開始するには、[ブランディングエディタを起動] を選択します。

トピック

- [ブランディングエディタによるマネージドログインのカスタマイズ](#)
- [ホステッド UI \(クラシック\) ブランディングのカスタマイズ](#)

ブランディングエディタによるマネージドログインのカスタマイズ

ブランディングエディタは、マネージドログインウェブページ用のビジュアルデザインおよび編集ツールです。Amazon Cognito コンソールに組み込まれています。ブランディングエディタでは、ログインページのプレビューから開始し、Quick Setup オプションまたは詳細ビューの詳細オプションに進むことができます。スタイルパラメータを変更してプレビューしたり、カスタム背景イメージとロゴを追加したりできます。ライトモードとダークモードを設定できます。



まず、ユーザープールまたはアプリケーションクライアントに適用できるスタイルを作成します。

ブランディングエディタの使用を開始するには

1. [\[ドメイン\]](#) タブからドメインを作成するか、既存のドメインを更新します。[ブランディングバージョン] で、[マネージドログイン] を使用するようにドメインを設定します。
2. 既存のアプリケーションクライアントスタイルがある場合は削除します。
 - a. [アプリケーションクライアント] メニューで、アプリケーションクライアントを選択します。
 - b. [マネージドログインスタイル] で、アプリクライアントに割り当てられたスタイルを選択します。
 - c. [スタイルを削除] を選択します。[Confirm] (確認) をクリックして、選択内容を確認します。
3. ユーザープールの [マネージドログイン] メニューに移動します。プロンプトに従って、マネージドログインを含む [機能プラン](#) を選択します (まだ選択していない場合)。変更を行わずにブランディングエディタをチェックアウトする場合は、[この機能をプレビュー] を選択することもできます。
4. [スタイル] で、[スタイルを作成] を選択します。
5. スタイルを割り当てるアプリケーションクライアントを選択し、[作成] を選択します。新しいアプリケーションクライアントを作成することもできます。
6. Amazon Cognito コンソールでブランディングエディタを起動します。

7. 編集を開始するタブを選択するか、[エディタを起動] を選択して [\[Quick Setup\]](#) に移動します。以下のタブが利用可能です。

プレビュー

マネージドログインページで現在の選択内容がどのように表示されるかを確認します。

基盤

全体的なテーマを設定し、外部 ID プロバイダーへのリンクとスタイルフォームフィールドを設定します。

コンポーネント

ヘッダー、フッター、個々の UI 要素のスタイルを設定します。

8. 初期設定を選択するには、Quick Setup に移動します。[設定カテゴリの変更] を選択し、[Quick Setup] を選択します。[続行] を選択すると、ブランディングエディタが起動し、設定する一連の基本オプションが表示されます。

テキストとローカリゼーション

ブランディングエディタでテキストを変更またはローカライズすることはできません。代わりに、ユーザーに配信する URL に lang クエリパラメータを追加します。このパラメータにより、マネージドログインページは利用可能な言語の 1 つにローカライズされます。詳細については、「[マネージドログインのローカリゼーション](#)」を参照してください。

Quick Setup

[ブランディングエディタを起動] ボタンを選択すると、マネージドログイン設定のビジュアルエディタがロードされ、さまざまな主要カスタマイズオプションから選択できます。選択するに従って、Amazon Cognito はマネージドログインの変更をプレビューウィンドウに反映します。詳細設定メニューに戻るには、[設定カテゴリの変更] ボタンを選択します。

全体的な外観と操作感をどのようにするか？

マネージドログインの基本的なテーマ設定を構成します。

Display mode (表示モード)

マネージドログインのライトモード、ダークモード、またはアダプティブエクスペリエンスを選択します。Amazon Cognito がマネージドログインをレンダリングすると、アダプティブ設

定はユーザーのブラウザ設定に従います。ブラウザアダプティブモードを選択すると、ライトモードとダークモードに異なる色とロゴ画像を選択できます。

[Spacing] (間隔)

ページの要素間のデフォルト間隔を設定します。

境界の半径

要素の外側の境界について、丸みの深さを設定します。

ページの背景をどのようにするか？

背景タイプ

[画像を表示] チェックボックスは、背景画像を使用するか、単色の背景を設定するかを示します。

1. 画像を使用するには、[画像を表示] を選択し、ライトモードとダークモードの背景画像を選択します。画像でカバーされない背景領域には、ダークモードとライトモードの [ページの背景色] を設定することもできます。
2. 背景に単色のみを使用するには、[画像を表示] の選択を解除し、ライトモードとダークモードの [ページの背景色] を選択します。

フォームの外観をどのようにするか？

マネージドログインのフォーム要素の設定を行います。フォーム項目の例として、ログインプロンプトとコードプロンプトがあります。

水平位置合わせ

フォームフィールドの水平位置合わせを設定します。

フォームのロゴ

ロゴ画像の配置を設定します。

ロゴ画像

ライトモードとダークモードのフォーム要素に含めるロゴ画像ファイルを選択します。画像をアップロードするには、[ロゴ画像] ドロップダウンを選択し、[新しいアセットを追加] を選択してロゴファイルを追加します。

主なブランドカラー

ライトモードとダークモードのテーマ色を設定します。この色は、プライマリとして分類されたすべての要素に背景色として適用されます。

ヘッダーの外観をどのようにするか？

マネージドログインページにヘッダーを含めるかどうかを選択します。ヘッダーにはロゴ画像を含めることができます。

ヘッダーのロゴ

ヘッダー内のロゴ画像の位置を設定します。

ロゴ画像

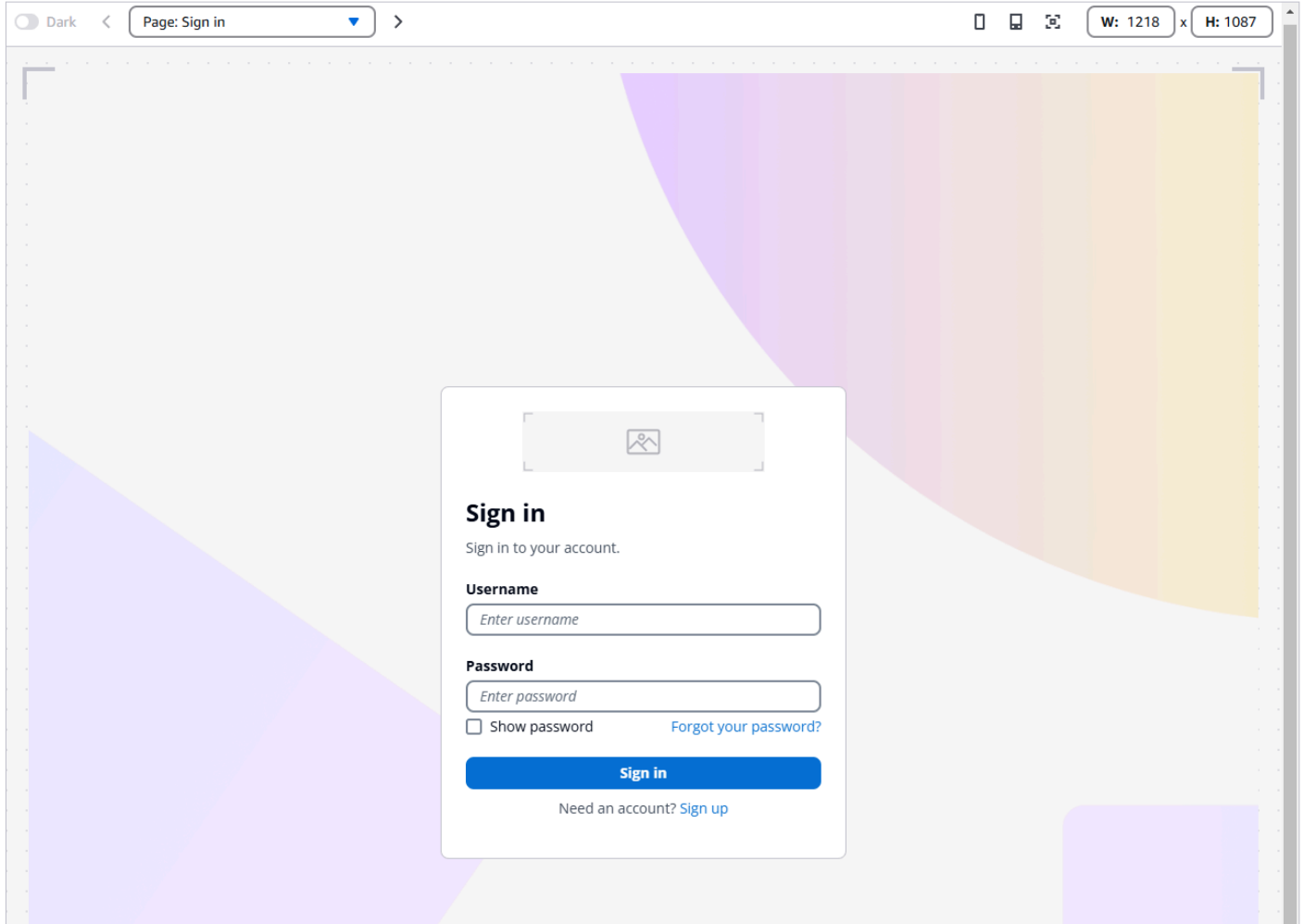
ヘッダーに含めるロゴの位置とロゴ画像ファイルを選択します。画像をアップロードするには、[ロゴ画像] ドロップダウンを選択し、[新しいアセットを追加] を選択してロゴファイルを追加します。

ヘッダーの背景色

ヘッダーの背景にライトモードとダークモードの色を設定します。

詳細設定

詳細設定ビューでは、[基盤] と [コンポーネント] の個々のコンポーネントを変更できます。[プレビュー] タブには、現在のコンテキストにおけるマネージドログインのプレビューとカスタマイズが表示されます。

[Amazon Cognito](#) > [User pools](#) > [User pool -](#) > [Managed login](#) > [Style:](#)**Style:** [Info](#)[Delete style](#)[Launch branding designer](#)**General information** [Info](#)**Assigned app client**
[My web app -](#) **Branding customizations**
[Cognito default settings](#)**Last customized time**
November 11, 2024 at 11:19 PST[Preview](#)[Foundation](#)[Components](#)

コンポーネントのビジュアルエディタに移動するには、コンポーネントのタイトルにある [編集] アイコンを選択します。テーマスタジオエディタで、[設定カテゴリを変更] ボタンを使用してコンポーネントを切り替えることができます。

基盤

アプリスタイル

マネージドログイン設定の基本を構成します。このカテゴリには、全体的なテーマ、テキスト間隔、ページのヘッダーとフッターの設定があります。

Display mode (表示モード)

マネージドログインページのライトモード、ダークモード、またはアダプティブエクスペリエンスを選択します。ブラウザアダプティブモードを選択すると、ライトモードとダークモードに異なる色とロゴ画像を選択できます。

[Spacing] (間隔)

ページの要素間のデフォルト間隔を設定します。

認証動作

ユーザーを外部 ID プロバイダー (IdP) に接続するボタンのスタイルを設定します。このセクションには、マネージドログインでユーザーに E メールアドレスの入力を求め、それを [SAML ID プロバイダー識別子](#)と照合するための [ドメイン検索入力] オプションが含まれています。

フォーム動作

入力フォームのスタイル (入力の配置、色、要素の位置合わせ) を設定します。

コンポーネント

ボタン

Amazon Cognito がマネージドログインページでレンダリングするボタンのスタイル。

ディバイダー

入力フォームや外部プロバイダーのサインインセレクターなど、マネージドログイン要素間の分割線と境界のスタイル。

ドロップダウン

ドロップダウンメニューのスタイル。

ファビコン

Amazon Cognito がタブとブックマークアイコンに提供する画像のスタイル。

フォーカスリング

現在選択されている入力を示すハイライトのスタイル。

フォームコンテナ

フォームを囲む要素のスタイル。

グローバルフッター

Amazon Cognito がマネージドログインページの下部に表示するフッターのスタイル。

グローバルヘッダー

Amazon Cognito がマネージドログインページの上部に表示するヘッダーのスタイル。

表示

エラーメッセージと成功メッセージのスタイル。

オプションコントロール

チェックボックス、複数選択、その他の入力プロンプトのスタイル。

ページ背景

マネージドログインの全体的な背景のスタイル。

入力

フォームフィールド入力プロンプトのスタイル。

Link

マネージドログインページのハイパーリンクのスタイル。

ページのテキスト

ページ内テキストのスタイル。

フィールドのテキスト

フォーム入力の周囲のテキストのスタイル。

マネージドログインのブランディングの API および SDK オペレーション

API オペレーション [CreateManagedLoginBranding](#) および [UpdateManagedLoginBranding](#) を使用して、マネージドログインスタイルにブランディングを適用することもできます。これらのオペレーションは、別のアプリケーションクライアントやユーザープールのブランディングス

スタイルと同一またはわずかに変更したバージョンを作成するのに最適です。API オペレーション [DescribeManagedLoginBranding](#) を使用して、既存のスタイルのマネージドログインブランディングをクエリし、必要に応じて出力を変更して別のリソースに適用します。

UpdateManagedLoginBranding オペレーションでは、スタイルを適用する先のアプリケーションクライアントは変更されません。アプリケーションクライアントに割り当てられている既存のスタイルのみが更新されます。アプリケーションクライアントのスタイルを完全に置き換えるには、[DeleteManagedLoginBranding](#) を使用して既存のスタイルを削除し、CreateManagedLoginBranding を使用して新しいスタイルを割り当てます。Amazon Cognito コンソールでも同様です。既存のスタイルを削除して新しいスタイルを作成する必要があります。

API または SDK リクエストでマネージドログインのブランディングを設定するには、Document データ型に変換された JSON ファイルに設定を埋め込む必要があります。以下は、追加できる画像と、ブランディングスタイルを設定するリクエストをプログラムで生成するためのガイダンスです。

画像アセット

[CreateManagedLoginBranding](#) と [UpdateManagedLoginBranding](#) には Assets パラメータが含まれています。このパラメータは、base64 でエンコードされたバイナリ形式の画像ファイルの配列です。

Note

ブランディングスタイルを作成または更新するプログラムリクエストは、リクエストサイズが 2 MB 以下である必要があります。リクエスト内のアセットによっては、この制限を超える場合があります。この場合、リクエストを複数の UpdateManagedLoginBranding リクエストに分割し、最大リクエストサイズを超えないパラメータのグループにします。これらのリクエストでは、指定していないパラメータがデフォルトに設定されることはないため、既存の設定に影響を与えずに、部分的なリクエストを送信できます。

一部のアセットは、送信できるファイルタイプに制限があります。

アセット	許可されるファイル拡張子
FAVICON_ICO	ico
FAVICON_SVG	svg

アセット	許可されるファイル拡張子
EMAIL_GRAPHIC	png、svg、jpeg
SMS_GRAPHIC	png、svg、jpeg
AUTH_APP_GRAPHIC	png、svg、jpeg
PASSWORD_GRAPHIC	png、svg、jpeg
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

SVG タイプのファイルは、以下の属性と要素をサポートしています。

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,
patterntransform, 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
```

マネージドログインのブランディングオペレーション用のツール

Amazon Cognito は、マネージドログインのブランディング設定オブジェクト用のファイルを [JSON スキーマ形式](#) で管理します。ブランディングスタイルをプログラムで更新する方法は、次のとおりです。

ユーザープール API でブランディングを更新するには

1. Amazon Cognito コンソールで、ユーザープールの [マネージドログイン] メニューからマネージドログインのデフォルトのブランディングスタイルを作成します。それをアプリケーションクライアントに割り当てます。
2. スタイルを作成したアプリケーションクライアントの ID (例: 1example23456789) を記録します。

3. ReturnMergedResources を true に設定して [DescribeManagedLoginBrandingByClient](#) API リクエストを使用し、ブランディングスタイルの設定を取得します。 リクエスト本文の例を次に示します。

```
{
  "ClientId": "lexample23456789",
  "ReturnMergedResources": true,
  "UserPoolId": "us-east-1_EXAMPLE"
}
```

4. DescribeManagedLoginBrandingByClient の出力をカスタマイズして変更します。
 - a. レスポンス本文は、作成および更新オペレーションの構文に含まれない ManagedLoginBranding 要素で囲まれています。この JSON オブジェクトの最上位レベルを削除します。
 - b. 画像を置き換えるには、Bytes 値を、各画像ファイルの Base64 エンコードされたバイナリデータに置き換えます。
 - c. 設定を更新するには、Settings オブジェクトの出力を変更し、次のリクエストに含めます。Amazon Cognito は、API レスポンスで受け取ったスキーマに存在しない Settings オブジェクトの値を無視します。
5. 更新されたレスポンス本文を [CreateManagedLoginBranding](#) または [UpdateManagedLoginBranding](#) リクエストで使用します。このリクエストのサイズが 2 MB を超える場合は、複数のリクエストに分割します。これらのオペレーションは、特に指定しない限り、元の設定が未変更のままである PATCH モデルで機能します。

ホステッド UI (クラシック) ブランディングのカスタマイズ

AWS マネジメントコンソール、AWS CLI または API を使用して、ホストされた UI の従来のカスタマイズ設定を指定できます。アプリに表示するカスタムのロゴイメージをアップロードできます。UI の外観と操作感にカスケーディングスタイルシート (CSS) オプションをいくつか適用することもできます。

UI のデフォルトをカスタマイズし、個々の[アプリケーションクライアント](#)を特定の設定で上書きできます。Amazon Cognito は、クライアントレベルの設定を持たないすべてのアプリケーションクライアントにデフォルト設定を適用します。

Amazon Cognito コンソールおよび API リクエストでは、UI のカスタマイズを設定するリクエストのサイズが 135 KB を超えることはできません。まれに、リクエストヘッダー、CSS ファイル、ロ

ゴの合計が 135 KB を超えることがあります。Amazon Cognito はイメージファイルを Base64 にエンコードします。これにより、100 KB のイメージのサイズが 130 KB に増え、リクエストヘッダーと CSS 用に 5 KB が残ります。リクエストが大きすぎる場合、AWS マネジメントコンソールまたは SetUICustomization API リクエストは `request parameters too large` エラーを返します。ロゴイメージを 100 KB 以下、CSS ファイルを 3 KB 以下に調整します。CSS とロゴのカスタマイズを個別に設定することはできません。

Note

UI をカスタマイズするには、ユーザープールのドメインを設定する必要があります。

クラシックブランディングでのカスタムロゴの指定

Amazon Cognito は、[ログインエンドポイント](#) の入力フィールドの上の中央にカスタムロゴを配置します。

ホストされているカスタムの UI ロゴには、350 x 178 ピクセルまで拡大できる PNG、JPG、または JPEG ファイルを選択します。ロゴファイルのサイズは 100 KB 以下で、Amazon Cognito が Base64 にエンコードした後は 130 KB 以下にしてください。API [SetUICustomization](#) で `ImageFile` を設定するには、ファイルを Base64-encoded されたテキスト文字列に変換するか、でファイルパス AWS CLI を指定して Amazon Cognito にエンコードさせます。

クラシックブランディングでの CSS カスタマイズの指定

ホストされたアプリページの CSS をカスタマイズできます。ただし、以下の制限があります。

- 以下の CSS クラス名を使用できます。
 - `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
- プロパティ値は HTML を含めることができます。ただし、@import,@supports,@page, または@mediaステートメント、または Javascript の値を除きます。

次の CSS プロパティをカスタマイズできます。

Labels

- [font-weight] は 100 ~ 900 までの 100 の倍数です。
- [color] はテキストの色です。 [有効な CSS カラー値](#)である必要があります。

入力フィールド

- [width] は含まれるブロックの幅に対してパーセンテージで表したものです。
- [height] は入力フィールドの高さ (ピクセル (px)) です。
- [color] はテキストの色です。任意の標準 CSS 色値を使用できます。
- [background-color] は、入力フィールドの背景色です。任意の標準 CSS 色値を使用できます。
- [border] は、アプリウィンドウの境界の幅、透明度、色を指定する標準 CSS 値です。幅は 1 px から 100 px までの任意の値を使用できます。透明度は solid または none です。色は任意の標準色値を使用できます。

テキストの説明

- [padding-top] は、説明欄の上のパディング量です。
- [padding-bottom] は、説明欄の下のパディング量です。
- [display] には block または inline のいずれかを設定できます。
- [font-size] はテキストの説明のフォントサイズです。
- [color] はテキストの色です。 [有効な CSS カラー値](#)である必要があります。

送信ボタン

- [font-size] はボタンテキストのフォントサイズです。
- [font-weight] はボタンテキストのフォントウェイトです。bold、italic、または normal です。
- [margin] はボタンの上下左右のマージンサイズを示す 4 つの値の文字列です。
- [font-size] はテキストの説明のフォントサイズです。
- [width] はボタンテキストのブロックに対する幅 (パーセント) です。
- [height] はボタンの高さ (ピクセル (px)) です。
- [color] はボタンの色です。任意の標準 CSS 色値を使用できます。
- [background-color] は、ボタンの背景色です。任意の標準色値を使用できます。

バナー

- [padding] はバナーの上下左右のパディングサイズを示す 4 つの値の文字列です。
- [background-color] は、バナーの背景色です。任意の標準 CSS 色値を使用できます。

送信ボタンのホバー

- [color] は、ボタンにカーソルを合わせたときのボタンの前景色です。任意の標準 CSS 色値を使用できます。
- [background-color] は、ボタンにカーソルを合わせたときのボタンの背景色です。任意の標準 CSS 色値を使用できます。

ID プロバイダーボタンのホバー

- [color] は、ボタンにカーソルを合わせたときのボタンの前景色です。任意の標準 CSS 色値を使用できます。
- [background-color] は、ボタンにカーソルを合わせたときのボタンの背景色です。任意の標準 CSS 色値を使用できます。

パスワードのチェックが有効ではありません

- [color] は "Password check not valid" メッセージのテキスト色です。任意の標準 CSS 色値を使用できます。

背景

- [background-color] は、アプリウィンドウの背景色です。任意の標準 CSS 色値を使用できます。

エラーメッセージ

- [margin] は上下左右のマージンサイズを示す 4 つの値の文字列です。
- [padding] はパディングサイズです。

- [font-size] はフォントサイズです。
- [width] はエラーメッセージの幅に含まれるブロックに対するパーセンテージで表したものです。
- [background] は、エラーメッセージの背景色です。任意の標準 CSS 色値を使用できます。
- [border] は境界の幅、透明度、色を指定する 3 つの値の文字列です。
- [color] はエラーメッセージのテキスト色です。任意の標準 CSS 色値を使用できます。
- [box-sizing] は含める必要があるサイズのプロパティ (幅と高さ) をブラウザに指示するために使用されます。

ID プロバイダーボタン

- [height] はボタンの高さ (ピクセル (px)) です。
- [width] はボタンテキストの幅に含まれるブロックに対するパーセンテージで示したものです。
- [text-align] はテキストの整列設定です。left、right または center のいずれかを設定できます。
- [margin-bottom] は下側マージンの設定です。
- [color] はボタンの色です。任意の標準 CSS 色値を使用できます。
- [background-color] は、ボタンの背景色です。任意の標準 CSS 色値を使用できます。
- [border-color] は、ボタンの境界の色です。任意の標準 CSS 色値を使用できます。

ID プロバイダーの説明

- [padding-top] は、説明欄の上のパディング量です。
- [padding-bottom] は、説明欄の下のパディング量です。
- [display] には block または inline のいずれかを設定できます。
- [font-size] は説明のフォントサイズです。
- color は、IdP セクションヘッダーのテキスト色です。例: 企業 ID でサインイン。 [有効な CSS カラー値](#)である必要があります。

法務関連のテキスト

- [color] はテキストの色です。任意の標準 CSS 色値を使用できます。
- [font-size] はフォントサイズです。

Note

リーガルテキストをカスタマイズする際は、サインインページのソーシャル ID プロバイダーに表示されるメッセージ「We won't post to any of your accounts without asking

first」(最初に確認せずにアカウントに投稿することはありません) をカスタマイズします。

ロゴ

- [max-width] は最大幅を含まれるブロックに対するパーセンテージで表したものです。
- [max-height] は最大高を含まれるブロックに対するパーセンテージで表したものです。
- background-color は、透明なセクションを持つロゴの背景の色です。[有効な CSS カラー値](#)である必要があります。

入力フィールドのフォーカス

- [border-color] は、入力フィールドの色です。任意の標準 CSS 色値を使用できます。
- [outline] は入力フィールドの境界線の幅 (ピクセル) です。

ソーシャルボタン

- [height] はボタンの高さ (ピクセル (px)) です。
- [text-align] はテキストの整列設定です。left、right または center のいずれかを設定できます。
- [width] はボタンテキストの幅を含まれるブロックに対するパーセンテージで示したものです。
- [margin-bottom] は下側マージンの設定です。

パスワードのチェックが有効です

- [color] は "Password check valid" メッセージのテキスト色です。任意の標準 CSS 色値を使用できます。

でクラシックブランドを使用してホストされた UI をカスタマイズする AWS マネジメントコンソール

を使用して AWS マネジメントコンソール、アプリの UI カスタマイズ設定を指定できます。

Note

以下の URL を、ユーザープールの指定を含めて作成し、ブラウザに入力することで、カスタマイズが反映されているホストされた UI を表示できます。 `https://<your_domain>/login?response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_uri>` 変更がコンソールに反映されるまで最大 1 分待ってからブラウザを最新表示にする必要がある場合があります。

ドメインは、[Domain] (ドメイン) の下の [App integration] (アプリケーション統合) タブに表示されます。アプリケーションクライアント ID およびコールバック URL は [App clients] (アプリケーションクライアント) に表示されています。

アプリ UI のカスタマイズ設定を指定する

1. [Amazon Cognito コンソール](#) にサインインします。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [\[ドメイン\]](#) タブからドメインを作成するか、既存のドメインを更新します。[ブランディングバージョン] で、[ホストされた UI (クラシック)] を使用するようにドメインを設定します。
4. [マネージドログイン] メニューを選択します。
5. すべてのアプリケーションクライアントの UI 設定をカスタマイズするには、[ホストされた UI 設定] で [スタイル] を参照し、[編集] を選択します。
6. 1 つのアプリケーションクライアントの UI 設定をカスタマイズするには、[アプリクライアント] メニューに移動し、変更するアプリケーションクライアントを選択します。次に [ホストされた UI (クラシック) スタイル] を参照し、[オーバーライド] を選択します。[Edit] (編集) を選択します。
7. 独自のロゴイメージファイルをアップロードするには、[Choose file] (ファイルを選択) または [Replace current file] (現在のファイルを置き換える) を選択します。
8. ホストされた UI CSS をカスタマイズするには、CSS template.css をダウンロードし、テンプレートをカスタマイズ値に変更します。ホストされた UI で使用できるのは、テンプレートに含まれるキーのみです。追加された CSS キーは UI に反映されません。CSS ファイルをカスタマイズしたら、[Choose file] (ファイルの選択) または [Replace current file] (現在のファイルを置き換える) を選択し、カスタム CSS ファイルをアップロードします。

ユーザープール API のクラシックブランドとを使用したホストされた UI のカスタマイズ AWS CLI

次のコマンドを使用して、ユーザープールのアプリ UI カスタマイズ設定を指定します。

ユーザープールの組み込みアプリ UI の UI カスタマイズ設定を取得するには、次の API オペレーションを使用します。

- AWS CLI: `aws cognito-idp get-ui-customization`
- AWS API: [GetUICustomization](#)

ユーザープールの組み込みアプリ UI の UI カスタマイズ設定を設定するには、次の API オペレーションを使用します。

- AWS CLI イメージファイルから: `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 Base64 バイナリテキストとしてエンコードされたイメージ: `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](#)

Lambda トリガーを使用したユーザープールワークフローのカスタマイズ

Amazon Cognito は AWS Lambda 関数と連携して、ユーザープールの認証動作を変更します。初回サインアップ前、認証完了後、およびその間のいくつかの段階で Lambda 関数を自動的に呼び出すようにユーザープールを設定できます。関数は、認証フローのデフォルトの動作を変更したり、ユーザープールやその他の AWS リソースを変更するための API リクエストを行ったり、外部システムと通信したりできます。Lambda 関数のコードはユーザー独自のものです。Amazon Cognito はイベントデータを関数に送信し、関数がデータを処理するのを待ちます。ほとんどの場合、セッションへの変更を示すレスポンスイベントが返されます。

リクエストイベントとレスポンスイベントのシステム内では、独自の認証チャレンジの導入、ユーザープールと別の ID ストア間でのユーザーの移行、メッセージのカスタマイズ、JSON ウェブトークン (JWT) の変更を行うことができます。

Lambda トリガーは、ユーザーがユーザープールでアクションを開始した後に Amazon Cognito がユーザーに配信するレスポンスをカスタマイズできます。例えば、他の方法では成功するはずのユーザーによるサインインを禁止できます。また、AWS 環境、外部 API、データベース、または ID ストアに対してランタイムオペレーションを実行することもできます。例えば、ユーザー移行のトリガーでは、外部アクションと Amazon Cognito の変更を組み合わせることができます。つまり、外部ディレクトリでユーザー情報を検索し、その外部情報に基づいて新しいユーザーに属性を設定できます。

Lambda トリガーをユーザープールに割り当てると、Amazon Cognito はデフォルトのフローを中断して関数から情報をリクエストします。Amazon Cognito は JSON イベントを生成し、それを関数に渡します。イベントには、ユーザーアカウントの作成、サインイン、パスワードのリセット、または属性の更新を求めるユーザーのリクエストに関する情報が含まれます。これで、関数がアクションを実行、またはイベントを変更せずに送り返すことができます。変更されていないイベントが返されると、イベントのデフォルトのアクションを続行するようにユーザープールに通知されます。例えば、サインアップ前トリガーは、PreSignUp_SignUp トリガーソースのユーザーを自動的に確認しますが、外部のユーザーと管理者が作成したユーザーの場合はイベントを変更せずに返すことができます。

次の表は、Lambda トリガーを使用してユーザープールオペレーションをカスタマイズする方法をまとめたものです。

ユーザープールフロー	オペレーション	説明
カスタム認証フロー	認証チャレンジの定義	カスタム認証フローでの次のチャレンジを決定する
	認証チャレンジの作成	カスタム認証フローでのチャレンジを作成する
	認証チャレンジレスポンスの確認	カスタム認証フローでのレスポンスが正しいかどうかを判断する
認証イベント	the section called “事前認証”	サインインリクエストを承認または拒否するカスタム検証
	the section called “認証後”	カスタム分析用のイベントをログに記録する
	the section called “トークン生成前”	トークンの主張を強化または制限する
フェデレーション	the section called “インバウンドフェデレーション”	Amazon Cognito ユーザープールでユーザーを作成または更新する前に、フェデレーションユーザー属性を変換します

ユーザープールフロー	オペレーション	説明
サインアップ	the section called “サインアップ前”	サインアップリクエストを承認または拒否するカスタム検証を実行する
	the section called “確認後”	カスタム分析用のカスタムウェルカムメッセージまたはイベントログ記録を作成する
	the section called “ユーザー移行”	既存のユーザーディレクトリからユーザープールにユーザーを移行する
メッセージ	the section called “カスタムメッセージ”	メッセージの高度なカスタマイズとローカライズを実行する
トークンの作成	the section called “トークン生成前”	ID トークンとアクセストークンの属性を追加または削除する
E メールと SMS のサードパーティープロバイダー	the section called “カスタム送信者”	サードパーティープロバイダーを使用して SMS メッセージと E メールメッセージを送信する

トピック

- [Lambda トリガーについて知っておくべきこと](#)
- [ユーザープールの Lambda トリガーを追加する](#)
- [ユーザープールの Lambda トリガーイベント](#)
- [ユーザープールの Lambda トリガーの一般的なパラメータ](#)
- [クライアントメタデータ](#)
- [Lambda トリガーへの API オペレーションの接続](#)
- [Lambda トリガーのユーザープールの機能オペレーションへの接続](#)
- [サインアップ前の Lambda トリガー](#)

- [確認後の Lambda トリガー](#)
- [認証前の Lambda トリガー](#)
- [認証後の Lambda トリガー](#)
- [インバウンドフェデレーション Lambda トリガー](#)
- [カスタム認証チャレンジの Lambda トリガー](#)
- [トークン生成前の Lambda トリガー](#)
- [ユーザー移行の Lambda トリガー](#)
- [カスタムメッセージの Lambda トリガー](#)
- [カスタム送信者の Lambda トリガー](#)

Lambda トリガーについて知っておくべきこと

Lambda 関数用のユーザープールを準備するときは、以下の点を考慮します。

- Amazon Cognito が Lambda トリガーに送信するイベントは、新しい機能に伴って変わる可能性があります。JSON 階層内のレスポンス要素とリクエスト要素の位置が変更されたり、要素名が追加されたりする場合があります。Lambda 関数では、このガイドで説明している入力要素のキーと値のペアを受け取ることが期待できますが、入力検証がより厳密になると、関数が失敗する可能性があります。
- Amazon Cognito が一部のトリガーに送信する複数のバージョンのイベントから 1 つを選択できます。バージョンによっては、Amazon Cognito 料金の変更を受け入れることが必要になる場合があります。料金の詳細については、「[Amazon Cognito の料金](#)」を参照してください。[トークン生成前の Lambda トリガー](#)でアクセストークンをカスタマイズするには、ライト以外の機能プランを使用してユーザープールを設定し、イベントバージョン 2 を使用するように Lambda トリガー設定を更新する必要があります。
- [カスタム送信者の Lambda トリガー](#)を除き、Amazon Cognito は Lambda 関数を同期的に呼び出します。Amazon Cognito が Lambda 関数を呼び出したら、5 秒以内に応答する必要があります。失敗し、呼び出しを再試行できる場合、Amazon Cognito は呼び出しを再試行します。試行が 3 回失敗すると、関数はタイムアウトします。この 5 秒のタイムアウト値を変更することはできません。詳細については、「AWS Lambda デベロッパーガイド」の「[Lambda プログラミングモデル](#)」を参照してください。

Amazon Cognito は、HTTP ステータスコード 500～599 の[呼び出しエラー](#)を返す関数呼び出しを再試行しません。これらのコードは、Lambda が関数を起動できない原因となる設定の問題を示しています。詳細については、「[エラー処理と自動再試行 AWS Lambda](#)」を参照してください。

- Lambda トリガー設定で関数バージョンを宣言することはできません。Amazon Cognito ユーザープールは、デフォルトで最新バージョンの関数を呼び出します。ただし、[CreateUserPool](#) または [UpdateUserPool](#) API リクエストにより、関数バージョンをエイリアスに関連付け、トリガーの LambdaArn をエイリアス ARN に設定できます。このオプションは、AWS マネジメントコンソールでは使用できません。エイリアスの詳細については、「AWS Lambda デベロッパーガイド」の「[Lambda 関数のエイリアス](#)」を参照してください。
- Lambda トリガーを削除する場合は、ユーザープール内の対応するトリガーを更新する必要があります。例えば、認証後トリガーを削除した場合は、ユーザープール内の対応する認証後トリガーを [なし] に設定する必要があります。
- Lambda 関数がリクエストとレスポンスのパラメータを Amazon Cognito に返さないか、エラーを返す場合、認証イベントは成功しません。関数にエラーを返して、ユーザーのサインアップ、認証、トークン生成、または Lambda トリガーを呼び出す認証フローのその他の段階を防ぐことができます。

マネージドログインは、Lambda トリガーが生成したエラーをサインインプロンプトの上にエラーテキストとして返します。Amazon Cognito ユーザープール API は、トリガーエラーを `[trigger] failed with error [error text from response]` 形式で返します。ベストプラクティスとして、Lambda 関数では、ユーザーに表示させるエラーのみを生成します。print() のような出力方法を使用して、機密情報やデバッグ情報を CloudWatch ログに記録します。例については、[サインアップ前の例:ユーザー名が 5 文字未満の場合にサインアップを拒否する](#)を参照してください。

- ユーザープールのトリガー AWS アカウント として、別の に Lambda 関数を追加できます。[CreateUserPool](#) および [UpdateUserPool](#) API オペレーション、または CloudFormation とで同等のものを使用して、クロスアカウントトリガーを追加する必要があります AWS CLI。にクロスアカウント関数を追加することはできません AWS マネジメントコンソール。
- Amazon Cognito コンソールに Lambda トリガーを追加すると、Amazon Cognito はユーザープールが関数を呼び出すことを許可するリソースベースのポリシーを、関数に追加します。クロスアカウント関数を含む Lambda トリガーを Amazon Cognito コンソールの外部で作成する場合は、Lambda 関数のリソースベースポリシーにアクセス許可を追加する必要があります。追加したアクセス権限では、Amazon Cognito はユーザープールの代わりに関数を呼び出すことを許可する必要があります。[Lambda コンソールから権限を追加](#)、または Lambda [AddPermission](#) API オペレーションを使用することができます。

Lambda リソースベースのポリシーの例

次の Lambda リソースベースのポリシーの例では、Lambda 関数を呼び出す Amazon Cognito の限定的な機能が付与されます。Amazon Cognito は、aws:SourceArn の条件でのユーザープールと

aws:SourceAccount 条件でのアカウントの両方に代わって関数を呼び出す場合にのみ、関数を呼び出すことができます。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "LambdaCognitoIdpTrust",
      "Effect": "Allow",
      "Principal": {
        "Service": "cognito-idp.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:MyFunction",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:cognito-idp:us-east-1:111122223333:userpool/us-east-1_EXAMPLE"
        }
      }
    }
  ]
}
```

ユーザープールの Lambda トリガーを追加する

コンソールを使用してユーザープールの Lambda トリガーを追加する

1. [Lambda コンソール](#)を使用して Lambda 関数を作成します。Lambda 関数の詳細については、[AWS Lambda デベロッパーガイド](#)を参照してください。
2. [\[Amazon Cognito console\]](#) (Amazon Cognito コンソール) に移動し、[User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。

4. [拡張機能] メニューを選択し、[Lambda トリガー] を見つけます。
5. [Add a Lambda trigger] (Lambda トリガーの追加) を選択します。
6. カスタマイズする認証のステージに基づいて、Lambda トリガーカテゴリを選択します。
7. Lambda 関数の割り当てを選択し、ユーザープール AWS リージョン と同じ 内の関数を選択します。

Note

AWS Identity and Access Management (IAM) 認証情報に Lambda 関数を更新するアクセス許可がある場合、Amazon Cognito は Lambda リソーススペースのポリシーを追加します。このポリシーを使用すると、Amazon Cognito は選択した関数を呼び出すことができます。サインインした認証情報に十分な IAM アクセス権限がない場合は、リソーススペースのポリシーを個別に更新する必要があります。詳細については、「[the section called “主要事項”](#)」を参照してください。

8. [Save changes] (変更の保存) をクリックします。
9. Lambda コンソールで CloudWatch を使用して、Lambda 関数をログすることができます。詳細については、「[Accessing CloudWatch Logs for Lambda](#)」を参照してください。

ユーザープールの Lambda トリガーイベント

Amazon Cognito は Lambda 関数にイベント情報を渡します。Lambda 関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。関数が入カイベントを変更せずに返した場合、Amazon Cognito はデフォルトの動作を続行します。以下に示すのは、すべての Lambda トリガー入カイベントに共通するパラメータです。トリガー別のイベント構文については、このガイドのトリガー別のセクションでイベントスキーマを確認してください。

JSON

```
{
  "version": "string",
  "triggerSource": "string",
  "region": AWSRegion,
  "userPoolId": "string",
  "userName": "string",
  "callerContext":
    {
      "awsSdkVersion": "string",
```

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

ユーザープールの Lambda トリガーの一般的なパラメータ

バージョン

Lambda 関数のバージョン番号。

triggerSource

Lambda 関数をトリガーしたイベントの名前。各 triggerSource の説明については、「[Lambda トリガーのユーザープールの機能オペレーションへの接続](#)」を参照してください。

リージョン

AWSRegion インスタンス AWS リージョン としての。

userPoolId

ユーザープールの ID。

userName

現在のユーザーのユーザー名。

callerContext

リクエストとコード環境に関するメタデータ。これには、[awsSdkVersion] と [clientId] フィールドが含まれています。

awsSdkVersion

リクエストを生成した AWS SDK のバージョン。

clientId

ユーザープールアプリクライアントの ID。

request

ユーザーの API リクエストの詳細。以下のフィールドと、トリガーに固有のリクエストパラメータが含まれます。例えば、Amazon Cognito が事前認証トリガーに送信するイベントには `userNotFound` パラメータも含まれます。ユーザーが未登録のユーザー名でサインインしようとしたときに、このパラメータの値を処理してカスタムアクションを実行できます。

userAttributes

ユーザー属性の名前と値の 1 つ以上のキーと値のペア。例: "email":
"john@example.com"。

response

このパラメータには、元のリクエストの情報は含まれていません。Lambda 関数はイベント全体を Amazon Cognito に返し、返されるパラメータは `response` に追加する必要があります。関数に含めることができる返されるパラメータを確認するには、使用するトリガーのドキュメントを参照してください。

クライアントメタデータ

API オペレーションと [トークンエンドポイント](#) リクエストでは、Lambda トリガー関数にカスタムパラメータを送信できます。クライアントメタデータを使用すると、アプリケーションはリクエスト元の環境に関する追加情報を収集できます。クライアントメタデータを Lambda 関数に渡すと、関数は追加のデータを処理することで、認証フローのログ記録やカスタマイズに使用できます。クライアントメタデータは、JSON 形式のキーと値の文字列ペアであり、自分で選択して作成します。

クライアントメタデータのユースケース例

- サインアップ時に位置情報データを[サインアップ前トリガー](#)に渡し、不要な場所からのサインインを防止します。
- テナント ID データを[カスタムチャレンジトリガー](#)に渡し、さまざまなビジネスユニットのユーザーごとに異なるチャレンジを発行します。
- ユーザーのトークンを[トークン生成前トリガー](#)に渡し、M2M リクエストを代行した対象のプリンシパルのログを生成します。リクエスト例については、「[基本認可によるクライアント認証情報](#)」を参照してください。

クライアントメタデータをサインアップ前トリガーに渡す例を以下に示します。

SignUp request

次に示すのは、[SignUp](#) リクエストにより、Amazon Cognito がクライアントメタデータをサインアップ前トリガーに渡す例です。

```
POST HTTP/1.1
Host: cognito-idp.us-east-1.amazonaws.com
X-Amz-Date: 20230613T200059Z
Accept-Encoding: gzip, deflate, br
X-Amz-Target: AWSCognitoIdentityProviderService.SignUp
User-Agent: <UserAgentString>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
  Signature=<Signature>
Content-Length: <PayloadSizeBytes>

{
  "ClientId": "1example23456789",
  "Username": "mary_major",
  "Password": "<Password>",
  "SecretHash": "<Secret hash>",
  "ClientMetadata": {
    "IpAddress" : "192.0.2.252",
    "GeoLocation" : "Netherlands (Kingdom of the) [NL]"
  }
  "UserAttributes": [
    {
      "Name": "name",
      "Value": "Mary"
    },
    {
      "Name": "email",
      "Value": "mary_major@example.com"
    },
    {
      "Name": "phone_number",
      "Value": "+12065551212"
    }
  ],
}
```

Lambda trigger input event

サインアップ前関数へのリクエストは、次のような本文になります。

```
{
  "callerContext": {
    "awsSdkVersion": "aws-sdk-unknown-unknown",
    "clientId": "1example23456789"
  },
  "region": "us-west-2",
  "request": {
    "clientMetadata": {
      "GeoLocation": "Netherlands (Kingdom of the) [NL]",
      "IpAddress": "192.0.2.252"
    },
    "userAttributes": {
      "email": "mary_major@example.com",
      "name": "Mary",
      "phone_number": "+12065551212"
    },
    "validationData": null
  },
  "response": {
    "autoConfirmUser": false,
    "autoVerifyEmail": false,
    "autoVerifyPhone": false
  },
  "triggerSource": "PreSignUp_SignUp",
  "userName": "mary_major2",
  "userPoolId": "us-west-2_EXAMPLE",
  "version": "1"
}
```

Machine to Machine (M2M) クライアント認証情報のクライアントメタデータ

[クライアントメタデータ](#)は M2M リクエストで渡すことができます。クライアントメタデータは、[トークン生成前の Lambda トリガー](#)の結果に役立つユーザーまたはアプリケーション環境からの追加情報です。ユーザープリンシパルによる認証オペレーションでは、[AdminRespondToAuthChallenge](#) API リクエストおよび [RespondToAuthChallenge](#) API リクエストの本文で、クライアントメタデータをトークン生成前トリガーに渡すことができます。アプリケーションは、[トークンエンドポイント](#)への直接リクエストを使用して M2M のアクセストークン生成フローを実行するため、異なるモデルとなります。クライアント認証情報のトークンリクエストの POST 本文で、クライアントメタデータオブジェクトを URL エンコード (x-www-form-urlencoded) した `aws_client_metadata` パラメータを文字列に渡します。リクエスト例について

では、「[基本認可によるクライアント認証情報](#)」を参照してください。次に示すのは、キーと値のペア {"environment": "dev", "language": "en-US"} を渡すパラメータの例です。

```
aws_client_metadata=%7B%22environment%22%3A%20%22dev%22,%20%22language%22%3A%20%22en-US%22%7D
```

一時的なユーザー属性: **validationData**

一部の認証オペレーションには validationData パラメータもあります。このパラメータにより、Amazon Cognito が自動的に収集しない外部情報を、クライアントメタデータと同様に、Lambda トリガーに渡すことができます。検証データフィールドは、サインアップおよびサインインオペレーションにおいて、追加のユーザーコンテキストを Lambda 関数に渡すことを目的としています。[SignUp](#) と [AdminCreateUser](#) は、validationData を[サインアップ前トリガー](#)に渡します。[InitiateAuth](#) と [AdminInitiateAuth](#) は、API リクエスト本文の ClientMetadata を入カイベントの validationData として、[認証前](#)トリガーと[ユーザー移行](#)トリガーに渡します。

クライアントメタデータを渡す先の関数に API オペレーションをマッピングする方法については、トリガーソースに関する以降のセクションを参照してください。

Lambda トリガーへの API オペレーションの接続

以下のセクションでは、Amazon Cognito がユーザープールのアクティビティから呼び出す Lambda トリガーについて説明します。

アプリケーションが、Amazon Cognito ユーザープールの API、マネージドログイン、またはユーザープールのエンドポイントを介してユーザーをサインインさせると、Amazon Cognito はセッションコンテキストに基づいて Lambda 関数を呼び出します。Amazon Cognito ユーザープール API およびユーザープールの詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。次のセクションの表では、Amazon Cognito が関数を呼び出す原因となるイベントと、Amazon Cognito がリクエストに含める triggerSource 文字列について説明します。

トピック

- [Amazon Cognito API の Lambda トリガー](#)
- [マネージドログインにおける Amazon Cognito ローカルユーザー向けの Lambda トリガー](#)
- [フェデレーティッドユーザーの Lambda トリガー](#)

Amazon Cognito API の Lambda トリガー

次の表は、アプリがローカルユーザーを作成、サインイン、または更新するときに Amazon Cognito が呼び出すことができる Lambda トリガーのソース文字列を示しています。

Amazon Cognito API のローカルユーザートリガーソース

API オペレーション:	Lambda トリガー	トリガーソース
AdminCreateUser	サインアップ前	PreSignUp_AdminCreateUser
	トークン生成前	TokenGeneration_NewPasswordChallenge
	カスタムメッセージ	CustomMessage_AdminCreateUser
	カスタム E メール送信者	CustomEmailSender_AdminCreateUser
	カスタム SMS 送信者	CustomSMSSender_AdminCreateUser
SignUp	サインアップ前	PreSignUp_SignUp
	カスタムメッセージ	CustomMessage_SignUp
	カスタム E メール送信者	CustomEmailSender_SignUp
	カスタム SMS 送信者	CustomSMSSender_SignUp
ConfirmSignUp AdminConfirmSignUp	確認後	PostConfirmation_ConfirmSignUp
InitiateAuth AdminInitiateAuth	事前認証	PreAuthentication_Authentication

API オペレーション:	Lambda トリガー	トリガーソース
	認証後	PostAuthentication_Authentication
	認証チャレンジを定義する	DefineAuthChallenge_Authentication
	認証チャレンジを作成する	CreateAuthChallenge_Authentication
	認証チャレンジの検証。	VerifyAuthChallenge_Authentication
	トークン生成前	TokenGeneration_Authentication
		TokenGeneration_AuthenticateDevice
		TokenGeneration_RefreshTokens
	ユーザー移行	UserMigration_Authentication
	カスタムメッセージ	CustomMessage_Authentication
	カスタム E メール送信者	CustomEmailSender_AccountTakeOverNotification
		CustomEmailSender_Authentication
	カスタム SMS 送信者	CustomSMSSender_Authentication

API オペレーション:	Lambda トリガー	トリガーソース
RespondToAuthChallenge AdminRespondToAuthChallenge	認証後	PostAuthentication_Authentication
	認証チャレンジを定義する	DefineAuthChallenge_Authentication
	認証チャレンジを作成する	CreateAuthChallenge_Authentication
	認証チャレンジの検証。	VerifyAuthChallenge_Authentication
	トークン生成前	TokenGeneration_Authentication
		TokenGeneration_AuthenticateDevice
		TokenGeneration_RefreshTokens
	カスタムメッセージ	CustomMessage_Authentication
	カスタム E メール送信者	CustomEmailSender_AccountTakeOverNotification
		CustomEmailSender_Authentication
ForgotPassword	カスタム SMS 送信者	CustomSMSSender_Authentication
	ユーザー移行	UserMigration_ForgotPassword

API オペレーション:	Lambda トリガー	トリガーソース
	カスタムメッセージ	CustomMessage_ForgotPassword
	カスタム E メール送信者	CustomEmailSender_ForgotPassword
	カスタム SMS 送信者	CustomSMSSender_ForgotPassword
ConfirmForgotPassword	確認後	PostConfirmation_ConfirmForgotPassword
UpdateUserAttributes AdminUpdateUserAttributes	カスタムメッセージ	CustomMessage_UpdateUserAttribute
	カスタム E メール送信者	CustomEmailSender_UpdateUserAttribute
	カスタム SMS 送信者	CustomSMSSender_UpdateUserAttribute
VerifyUserAttributes	カスタムメッセージ	CustomMessage_VerifyUserAttribute
	カスタム E メール送信者	CustomEmailSender_VerifyUserAttribute
	カスタム SMS 送信者	CustomSMSSender_VerifyUserAttribute
GetTokensFromRefreshToken	トークン生成前	TokenGeneration_Authentication

マネージドログインにおける Amazon Cognito ローカルユーザー向けの Lambda トリガー

次の表は、ローカルユーザーがマネージドログインを使用してユーザープールにサインインするときに Amazon Cognito が呼び出せる Lambda トリガーのソース文字列を示しています。

マネージドログインにおけるローカルユーザーのトリガーソース

マネージドログイン URI	Lambda トリガー	トリガーソース
/signup	サインアップ前	PreSignUp_SignUp
	カスタムメッセージ	CustomMessage_SignUp
	カスタム E メール送信者	CustomEmailSender_SignUp
	カスタム SMS 送信者	CustomSMSSender_SignUp
/confirmuser	確認後	PostConfirmation_ConfirmSignUp
/login	事前認証	PreAuthentication_Authentication
	トークン生成前	TokenGeneration_Authentication
		TokenGeneration_AuthenticateDevice
		TokenGeneration_RefreshTokens
	ユーザー移行	UserMigration_Authentication
	カスタムメッセージ	CustomMessage_Authentication

マネージドログイン URI	Lambda トリガー	トリガーソース
/forgotpassword	カスタム E メール送信者	CustomEmailSender_AccountTakeOverNotification CustomEmailSender_Authentication
	カスタム SMS 送信者	CustomSMSSender_Authentication
	ユーザー移行	UserMigration_ForgotPassword
	カスタムメッセージ	CustomMessage_ForgotPassword
	カスタム E メール送信者	CustomEmailSender_ForgotPassword
	カスタム SMS 送信者	CustomSMSSender_ForgotPassword
/confirmforgotpassword	確認後	PostConfirmation_ConfirmForgotPassword

フェデレーテッドユーザーの Lambda トリガー

次の Lambda トリガーを使用して、フェデレーションプロバイダーでサインインするユーザーのユーザープールのワークフローをカスタマイズできます。

Note

フェデレーションユーザーは、マネージドログインを使用してサインインできます。または、[認可エンドポイント](#) へのリクエストを生成して、ユーザーを ID プロバイダーのサイ

ンインページにサイレントにリダイレクトすることができます。Amazon Cognito ユーザープール API を使用してフェデレーションユーザーをサインインすることはできません。

フェデレーティッドユーザートリガーのソース

サインインイベント	Lambda トリガー	トリガーソース
初回サインイン	サインアップ前	PreSignUp_External Provider
	確認後	PostConfirmation_ConfirmSignUp
	トークン生成前	TokenGeneration_HostedAuth
その後のサインイン	事前認証	PreAuthentication_Authentication
	認証後	PostAuthentication_Authentication
	トークン生成前	TokenGeneration_HostedAuth

フェデレーティッドサインインは、ユーザープールの [カスタム認証チャレンジの Lambda トリガー](#)、[ユーザー移行の Lambda トリガー](#)、[カスタムメッセージの Lambda トリガー](#)、または [カスタム送信者の Lambda トリガー](#) を呼び出しません。

Lambda トリガーのユーザープールの機能オペレーションへの接続

各 Lambda トリガーは、ユーザープールで機能的な役割を果たします。例えば、トリガーを使用してサインアップフローを変更したり、カスタム認証チャレンジを追加したりできます。Amazon Cognito が Lambda 関数に送信するイベントには、その機能ロールを構成する複数のアクションのうちの 1 つが反映されます。例えば、Amazon Cognito は、ユーザーがサインアップしたとき、およびユーザーを作成したときに、事前サインアップトリガーを呼び出します。同じ機能的役割を持つ、これらのさまざまなケースには、それぞれ独自の triggerSource 値があります。Lambda 関数は、呼び出したオペレーションに基づいて受信イベントを異なる方法で処理できます。

また Amazon Cognito は、イベントがトリガーソースに対応するときに、割り当てられたすべての関数を呼び出します。例えば、ユーザー移行トリガーと事前認証トリガーを割り当てたユーザープールにユーザーがサインインすると、両方がアクティブになります。

サインアップ、確認、およびサインイン (認証) トリガー

トリガー	triggerSource 値	イベント
サインアップ前	PreSignUp_SignUp	サインアップ前。
サインアップ前	PreSignUp_AdminCreateUser	管理者が新しいユーザーを作成するときのサインアップ前
サインアップ前	PreSignUp_ExternalProvider	外部 ID プロバイダーのサインアップ前。
確認後	PostConfirmation_ConfirmSignUp	サインアップの確認後。
確認後	PostConfirmation_ConfirmForgotPassword	パスワードを忘れた場合の確認後。
認証前	PreAuthentication_Authentication	認証前。
認証後	PostAuthentication_Authentication	認証後。

カスタム認証チャレンジトリガー

Trigger	triggerSource 値	イベント
認証チャレンジを定義する	DefineAuthChallenge_Authentication	認証チャレンジの定義。
認証チャレンジの作成	CreateAuthChallenge_Authentication	認証チャレンジの作成。

Trigger	triggerSource 値	イベント
認証チャレンジの検証。	VerifyAuthChallengeResponse_Authentication	認証チャレンジレスポンスの確認。

フェデレーショントリガー

Trigger	triggerSource 値	イベント
インバウンドフェデレーション	InboundFederation_ExternalProvider	インバウンドフェデレーション。

トークン生成前トリガー

Trigger	triggerSource 値	イベント
トークン生成前	TokenGeneration_HostedAuth	Amazon Cognito は、マネージドログインのサインインページからユーザーを認証します。
トークン生成前	TokenGeneration_Authentication	ユーザーの認証またはトークンの更新が完了しました。
トークン生成前	TokenGeneration_NewPasswordChallenge	管理者がユーザーを作成します。ユーザーが一時パスワードを変更する必要があるときに、Amazon Cognito は、これを呼び出します。
トークン生成前	TokenGeneration_AuthenticateDevice	ユーザーデバイスの認証の終了。
トークン生成前	TokenGeneration_RefreshTokens	ユーザーは、ID およびアクセストークンを更新しようとします。

移行ユーザートリガー

Trigger	triggerSource 値	イベント
ユーザー移行	UserMigration_Authentication	サインイン時のユーザー移行。
ユーザー移行	UserMigration_ForgotPassword	パスワードを忘れた場合のフ ロー実行時のユーザー移行

カスタムメッセージトリガー

Trigger	triggerSource 値	イベント
カスタムメッセージ	CustomMessage_SignUp	ユーザーがユーザープールにサインアップしたときのカスタムメッセージ。
カスタムメッセージ	CustomMessage_AdminCreateUser	管理者としてユーザーを作成し、Amazon Cognito から一時パスワードが送信されるときのカスタムメッセージ。
カスタムメッセージ	CustomMessage_ResendCode	既存のユーザーが新しい確認コードをリクエストしたときのカスタムメッセージ。
カスタムメッセージ	CustomMessage_ForgotPassword	ユーザーがパスワードのリセットを要求したときのカスタムメッセージ。
カスタムメッセージ	CustomMessage_UpdateUserAttribute	ユーザーが E メールアドレスまたは電話番号を変更し、Amazon Cognito が検証コードを送信したときのカスタムメッセージ。
カスタムメッセージ	CustomMessage_VerifyUserAttribute	ユーザーが E メールアドレスまたは電話番号を追加し

Trigger	triggerSource 値	イベント
		、Amazon Cognito が検証コードを送信したときのカスタムメッセージ。
カスタムメッセージ	CustomMessage_Authentication	SMS MFA を設定したユーザーがサインインしたときのカスタムメッセージ。

カスタム送信者トリガー

Trigger	triggerSource 値	イベント
カスタム送信者	CustomEmailSender_SignUp CustomSmsSender_SignUp	ユーザーがユーザープールにサインアップした場合。
カスタム送信者	CustomEmailSender_AdminCreateUser CustomSmsSender_AdminCreateUser	管理者としてユーザーを作成し、Amazon Cognito がユーザーに仮パスワードを送信した場合。
カスタム送信者	CustomEmailSender_ForgotPassword CustomSmsSender_ForgotPassword	ユーザーがパスワードのリセットをリクエストした場合。
カスタム送信者	CustomEmailSender_UpdateUserAttribute CustomSmsSender_UpdateUserAttribute	ユーザーが E メールアドレスまたは電話番号を変更し、Amazon Cognito が検証コードを送信した場合。

Trigger	triggerSource 値	イベント
カスタム送信者	CustomEmailSender_VerifyUserAttribute CustomSmsSender_VerifyUserAttribute	ユーザーが E メールアドレスまたは電話番号を追加し、Amazon Cognito が検証コードを送信した場合。
カスタム送信者	CustomEmailSender_Authentication CustomSmsSender_Authentication	SMS、E メール MFA、または OTP を設定したユーザーがサインインした場合。
カスタム送信者	CustomEmailSender_AccountTakeOverNotification	脅威保護設定により、ユーザーのサインイン試行に対して自動アクションが実行され、リスクレベルのアクションに通知が含まれている場合。

サインアップ前の Lambda トリガー

セルフサービスのサインアップオプションがあるユーザープールでサインアッププロセスをカスタマイズする場合があります。サインアップ前トリガーの一般的な用途は、新規ユーザーのカスタム分析と記録の実行、セキュリティとガバナンスの基準の適用、またはサードパーティー IdP から[統合されたユーザープロフィール](#)へのユーザーのリンクです。また、[検証と確認](#)を受ける必要がない信頼されたユーザーが存在する場合があります。

Amazon Cognito は、[ローカル](#)ユーザーまたは[フェデレーション](#)ユーザーの新規作成を完了する直前に、サインアップ前 Lambda 関数をアクティブ化します。この関数に送信したリクエストオブジェクトの userAttributes には、ローカルユーザーのサインアップ時に指定した属性、またはフェデレーションユーザーのプロバイダー属性から正常にマッピングされた属性が含まれています。ユーザープールは、[SignUp](#) によるセルフサービスのサインアップ時や信頼できる [ID プロバイダー](#) による初回サインイン時、および [AdminCreateUser](#) によるユーザー作成時に、このトリガーを呼び出します。サインアッププロセスの一環として、この関数を使用してカスタムロジックでのサインインイベントの分析や、新しいユーザーの変更または拒否ができます。

トピック

- [サインアップ前の Lambda トリガーのパラメータ](#)
- [サインアップ前の例: 登録済みドメインのユーザーを自動確認する](#)
- [サインアップ前の例: すべてのユーザーを自動確認して自動検証する](#)
- [サインアップ前の例: ユーザー名が 5 文字未満の場合にサインアップを拒否する](#)

サインアップ前の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

```
{
  "request": {
    "userAttributes": {
      "string": "string",
      . . .
    },
    "validationData": {
      "string": "string",
      . . .
    },
    "clientMetadata": {
      "string": "string",
      . . .
    }
  },
  "response": {
    "autoConfirmUser": "boolean",
    "autoVerifyPhone": "boolean",
    "autoVerifyEmail": "boolean"
  }
}
```

サインアップ前のリクエストパラメータ

userAttributes

ユーザー属性を表す 1 つ以上の名前 - 値ペア。属性名はキーです。

validationData

新しいユーザーの作成リクエストでアプリケーションから Amazon Cognito に渡したユーザー属性データを示す 1 つ以上のキーと値のペアです。この情報を [AdminCreateUser](#) API リクエストまたは [SignUp](#) API リクエストの ValidationData パラメータで Lambda 関数に送信します。

Amazon Cognito は、ValidationData データを、作成したユーザーの属性として設定しません。ValidationData は、サインアップ前 Lambda トリガーのためにユーザーが提供する一時的なユーザー情報です。

clientMetadata

サインアップ前のトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。このデータは、[AdminCreateUser](#)、[AdminRespondToAuthChallenge](#)、[ForgotPassword](#)、および [SignUp](#) の API アクションで ClientMetadata パラメータを使用することによって Lambda 関数に渡すことができます。

サインアップ前のレスポンスパラメータ

ユーザーを自動的に確認する場合は、レスポンスで autoConfirmUser を true に設定できます。autoVerifyEmail を true に設定してユーザーの E メールを自動検証できます。autoVerifyPhone を true に設定してユーザーの電話番号を自動検証できます。

Note

サインアップ前 Lambda 関数が AdminCreateUser API によってトリガーされる場合、レスポンスパラメータの autoVerifyPhone、autoVerifyEmail、および autoConfirmUser は Amazon Cognito によって無視されます。

autoConfirmUser

ユーザーを自動的に確認する場合は true に、それ以外の場合は false に設定します。

autoVerifyEmail

true に設定すると、サインアップしたユーザーの E メールアドレスを検証済みとして設定します。それ以外の場合は false です。autoVerifyEmail が true に設定されている場合、email 属性は有効な null 以外の値である必要があります。それ以外の場合はエラーが発生し、ユーザーがサインアップを完了できません。

email 属性がエイリアスとして選択されている場合、autoVerifyEmail が設定されていると、ユーザーの E メールアドレスのエイリアスが自動的に作成されます。その E メールアドレスのエイリアスが既に存在している場合は、エイリアスは新規ユーザーに移動され、以前のユーザーの E メールアドレスは未検証としてマークされます。詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

autoVerifyPhone

true に設定すると、サインアップしたユーザーの電話番号を検証済みとして設定します。それ以外の場合は false です。autoVerifyPhone が true に設定されている場合、phone_number 属性は有効な null 以外の値である必要があります。それ以外の場合はエラーが発生し、ユーザーがサインアップを完了できません。

phone_number 属性がエイリアスとして選択されている場合、autoVerifyPhone が設定されていると、ユーザーの電話番号のエイリアスが自動的に作成されます。その電話番号のエイリアスが既に存在している場合、エイリアスは新規ユーザーに移動され、以前のユーザーの電話番号は未検証としてマークされます。詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

サインアップ前の例: 登録済みドメインのユーザーを自動確認する

これは Lambda トリガーコードの例です。サインアップ前トリガーは、Amazon Cognito がサインアップリクエストを処理する直前に呼び出されます。これは、カスタム属性の [custom:domain] を使用して特定の E メールドメインからの新しいユーザーを自動的に確認します。このカスタムドメインに属していない新しいユーザーは、ユーザープールには追加されますが、自動的に確認されません。

Node.js

```
export const handler = async (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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

```
{
  "request": {
    "userAttributes": {
```

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

サインアップ前の例: すべてのユーザーを自動確認して自動検証する

次の例では、すべてのユーザーを確認し、ユーザーの email 属性と phone_number 属性を検証済みとして設定します (属性が存在する場合)。また、エイリアシングが有効になっている場合は、自動検証が設定されていると、phone_number および email にエイリアスが作成されます。

Note

同じ電話番号のエイリアスが既に存在している場合は、エイリアスは新規ユーザーに移動され、以前のユーザーの phone_number は未検証としてマークされます。E メールアドレスの場合も同様です。これを防ぐには、ユーザープール [ListUsers API](#) を使用して既存のユーザーの中に、新しいユーザーの電話番号や E メールアドレスをエイリアスとして既に使用しているユーザーがないかどうかを確認します。

Node.js

```
exports.handler = (event, context, callback) => {
  // Confirm the user
  event.response.autoConfirmUser = true;

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

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

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

```
};
```

Python

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

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

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

    # Return to Amazon Cognito
    return event
```

Amazon Cognito は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

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

サインアップ前の例:ユーザー名が 5 文字未満の場合にサインアップを拒否する

次の例は、サインアップリクエストのユーザー名の長さを調べます。次の例は、ユーザーが 5 文字未満の名前を入力した場合、エラーを返します。

Node.js

```
export const 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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

```
{
  "userName": "rroe",
  "response": {}
}
```

確認後の Lambda トリガー

Amazon Cognito は、サインアップしたユーザーがユーザーアカウントを確認した後にこのトリガーを呼び出します。投稿確認 Lambda 関数では、カスタムメッセージを送信したり、カスタム API リクエストを追加したりできます。たとえば、外部システムにクエリを実行して、ユーザーに追加の属性を入力できます。Amazon Cognito は、管理者認証情報を使用して作成したユーザーではなく、ユーザープールにサインアップしたユーザーに対してのみこのトリガーを呼び出します。

リクエストには、確認されたユーザーの現在の属性が含まれています。ユーザープールは、[ConfirmSignUp](#)、[AdminConfirmSignUp](#)、[ConfirmForgotPassword](#) に対して確認後の関数を呼び出します。このトリガーは、ユーザーが[マネージドログイン](#)でサインアップまたはパスワードのリセットを確認したときにも実行されます。

トピック

- [確認後の Lambda トリガーのパラメータ](#)
- [確認後の例](#)

確認後の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

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

確認後のリクエストパラメータ

userAttributes

ユーザー属性を表す 1 つ以上のキーバリューペア。

clientMetadata

確認後のトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。このデータは、[AdminConfirmSignUp](#)、[ConfirmForgotPassword](#)、[ConfirmSignUp](#)、および [SignUp](#) の API アクションで ClientMetadata パラメータを使用することによって Lambda 関数に渡すことができます。

確認後のレスポンスパラメータ

レスポンスで返される追加の情報は想定されていません。

確認後の例

この Lambda 関数の例では、Amazon SES を使用してユーザーに確認 E メールメッセージを送信します。詳細については、[Amazon Simple Email Service デベロッパーガイド](#)を参照してください。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

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

```
}
```

認証前の Lambda トリガー

Amazon Cognito は、ユーザーがサインインしようとするときにこのトリガーを呼び出し、準備アクションを実行するカスタム認証を作成できるようにします。たとえば、認証リクエストを拒否したり、外部システムへのセッションデータを記録したりできます。

Note

この Lambda トリガーは、ユーザープールのアプリケーションクライアントの `PreventUserExistenceErrors` 設定を `ENABLED` に指定していない限り、ユーザーが存在しない場合はアクティブになりません。既存の認証セッションを更新しても、このトリガーはアクティブになりません。

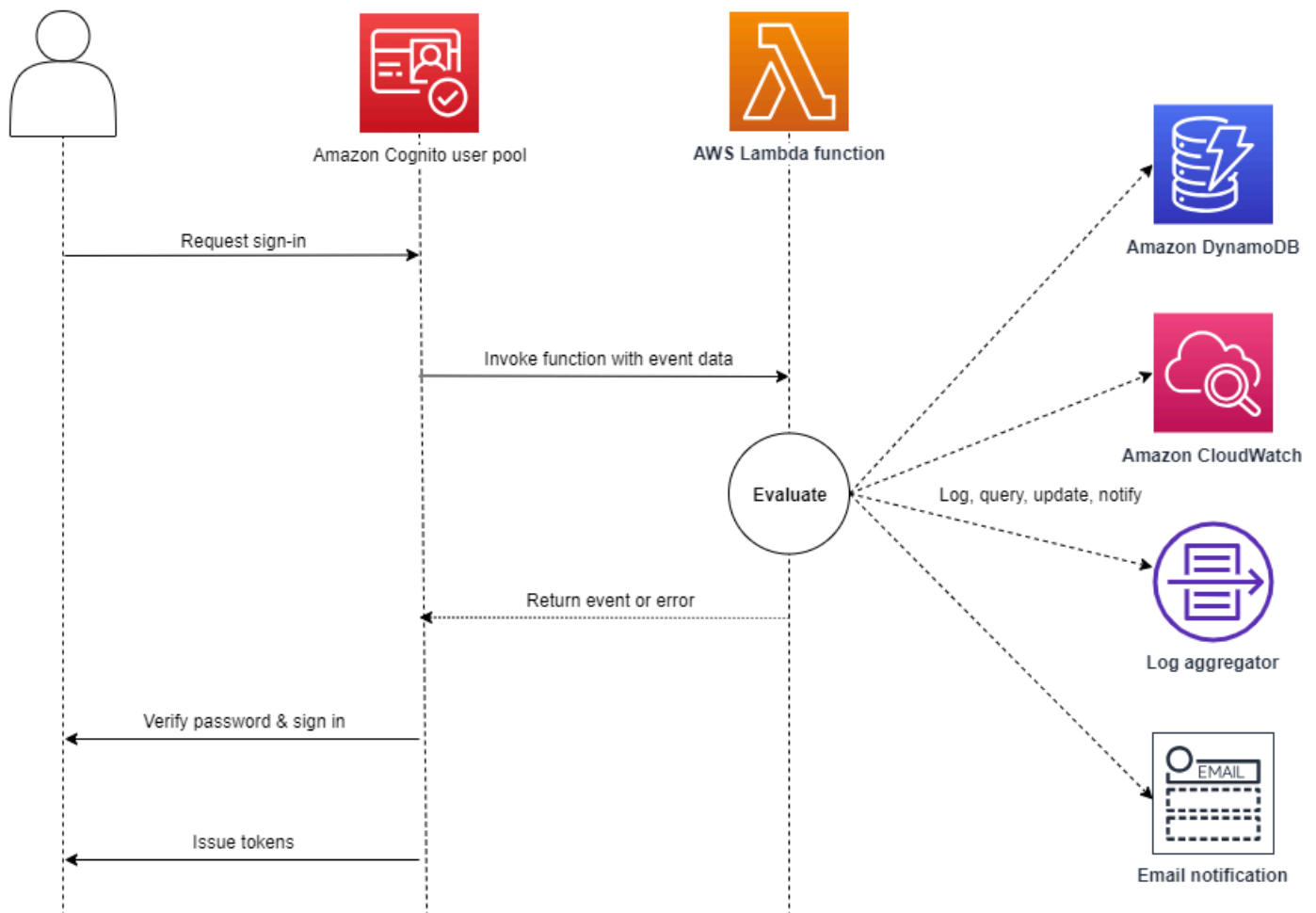
トピック

- [フローの概要](#)
- [認証前の Lambda トリガーのパラメータ](#)
- [事前認証の例](#)

フローの概要

Amazon Cognito pre authentication trigger

Evaluate and authorize user sign-in



リクエストには、アプリケーションからユーザープールの InitiateAuth および AdminInitiateAuth API オペレーションに渡す ClientMetadata 値のクライアント検証データが含まれます。

詳細については、「[認証セッションの例](#)」を参照してください。

認証前の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する [共通パラメータ](#) を組み合わせたものです。

JSON

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

認証前のリクエストパラメータ

userAttributes

ユーザー属性を表す 1 つ以上の名前 - 値ペア。

userNotFound

ユーザープールクライアントの `PreventUserExistenceErrors` を `ENABLED` に設定すると、Amazon Cognito はこのブール値を入力します。

validationData

ユーザーのサインインリクエストに検証データを含む 1 つ以上のキーバリューペア。このデータを Lambda 関数に渡すには、[InitiateAuth](#) および [AdminInitiateAuth](#) API アクションで `ClientMetadata` パラメータを使用します。

認証前のレスポンスパラメータ

Amazon Cognito は、関数がレスポンスで返す追加情報を処理しません。関数がエラーを返してサインイン試行を拒否したり、API オペレーションを使用してリソースをクエリしたり変更できます。

事前認証の例

このサンプル関数は、特定のアプリケーションからユーザープールにサインインされるのを防ぎます。事前認証の Lambda 関数は、ユーザーが既存のセッションを持っている場合は呼び出されないため、この関数はブロックするアプリケーション ID の新規セッションのみを防止します。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

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

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

認証後の Lambda トリガー

認証後トリガーは、ユーザーの認証フローを変更しません。Amazon Cognito は、認証が完了した後で、ユーザーがトークンを受け取る前に、この Lambda を呼び出します。次のサインインに反映されるログ記録やユーザープロファイルの調整など、認証イベントの後処理をカスタムで追加する場合は、認証後トリガーを追加します。

リクエスト本文を Amazon Cognito に返さない認証後 Lambda は、認証の完了に失敗する可能性があります。詳細については、「[Lambda トリガーについて知っておくべきこと](#)」を参照してください。

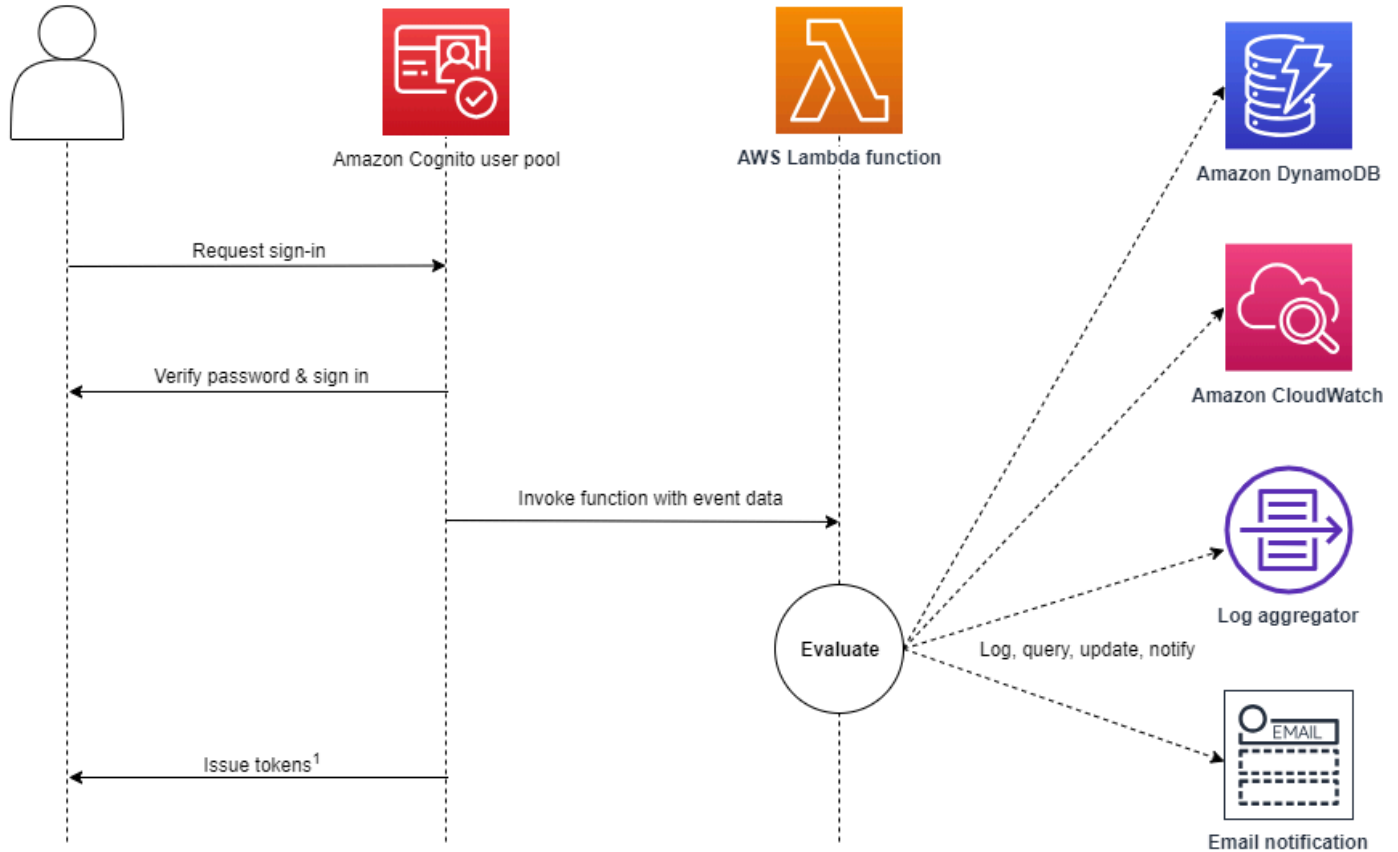
トピック

- [認証フローの概要](#)
- [認証後の Lambda トリガーのパラメータ](#)
- [認証後の例](#)

認証フローの概要

Amazon Cognito post authentication trigger

Report sign-in results



¹ This trigger doesn't have any effect on sign-in outcomes or token contents.

詳細については、「[認証セッションの例](#)」を参照してください。

認証後の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

```
{
  "request": {
    "userAttributes": {
```

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

認証後のリクエストパラメータ

newDeviceUsed

このフラグは、ユーザーが新しいデバイスにサインインしているかどうかを示します。Amazon Cognito は、ユーザープールの記憶済みデバイス値が Always または User Opt-In である場合にのみ、このフラグを設定します。

userAttributes

ユーザー属性を表す 1 つ以上の名前 - 値ペア。

clientMetadata

認証後のトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。このデータを Lambda 関数に渡すには、[AdminRespondToAuthChallenge](#) および [RespondToAuthChallenge](#) API アクションで ClientMetadata パラメータを使用します。Amazon Cognito は、認証後関数に渡すリクエストの [AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションの ClientMetadata パラメータからのデータを含めません。

認証後のレスポンスパラメータ

Amazon Cognito は、レスポンスに追加の返品情報を期待しません。関数では、API オペレーションを使用してリソースをクエリして変更したり、イベントメタデータを外部システムに記録することができます。

認証後の例

この認証後の Lambda 関数のサンプルは、正常に行われたサインインからのデータを CloudWatch Logs に送信します。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

```
{
  "triggerSource": "testTrigger",
  "userPoolId": "testPool",
  "userName": "testName",
```

```
"callerContext": {
  "clientId": "12345"
},
"response": {}
}
```

インバウンドフェデレーション Lambda トリガー

インバウンドフェデレーショントリガーは、外部 ID プロバイダーによる認証プロセス中にフェデレーションユーザー属性を変換します。ユーザーが設定された ID プロバイダーを介して認証する場合、このトリガーにより、認証プロセスでデータを傍受して変換することで外部 SAML プロバイダーと OIDC プロバイダーからのレスポンスを変更でき、Amazon Cognito ユーザープールがフェデレーティッドユーザーとその属性を処理する方法をプログラムで制御できます。

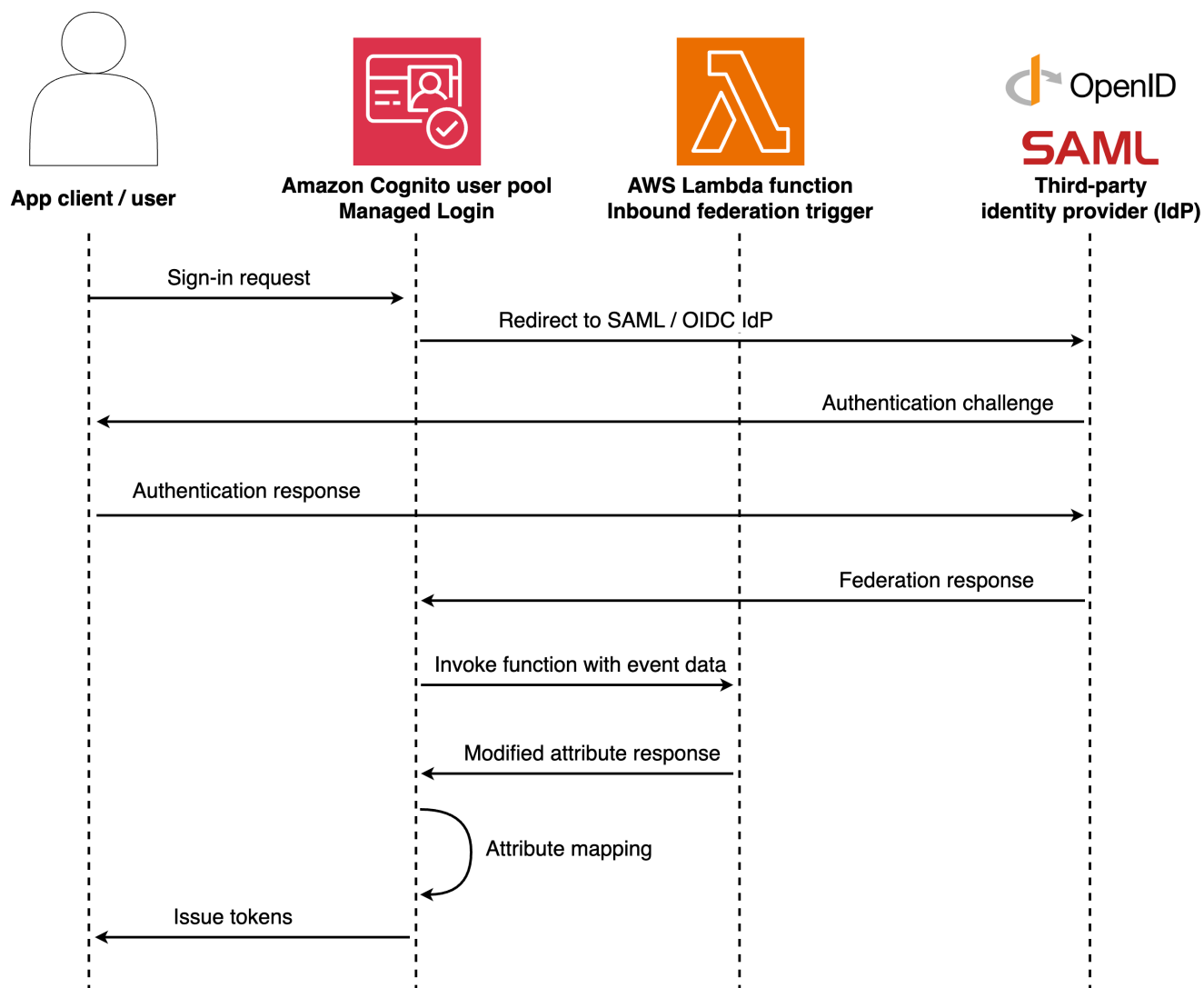
このトリガーを使用して、新しいユーザーを作成する前、または既存のフェデレーティッドユーザープロフィールを更新する前に、属性を追加、上書き、または抑制します。このトリガーは、raw ID プロバイダー属性を入力として受け取り、Amazon Cognito がユーザープロフィールに適用する変更された属性を返します。

トピック

- [フローの概要](#)
- [インバウンドフェデレーション Lambda トリガーパラメータ](#)
- [インバウンドフェデレーションの例: グループメンバーシップ管理](#)
- [インバウンドフェデレーションの例: 大きな属性を切り捨てる](#)
- [インバウンドフェデレーションの例: フェデレーションイベントのログ記録](#)

フローの概要

ユーザーが外部 ID プロバイダーで認証されると、Amazon Cognito はユーザープロフィールを作成または更新する前にインバウンドフェデレーショントリガーを呼び出します。トリガーは ID プロバイダーから raw 属性を受け取り、Amazon Cognito が保存する前に変換できます。このフローは、フェデレーションを通じて再度サインインする新しいフェデレーティッドユーザーと既存のユーザーの両方に発生します。



インバウンドフェデレーション Lambda トリガーパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する [共通パラメータ](#) を組み合わせたものです。

JSON

```
{
  "version": "string",
  "triggerSource": "InboundFederation_ExternalProvider",
  "region": AWSRegion,
  "userPoolId": "string",
  "userName": "string",
```

```
{
  "callerContext": {
    "awsSdkVersion": "string",
    "clientId": "string"
  },
  "request": {
    "providerName": "string",
    "providerType": "string",
    "attributes": {
      "tokenResponse": {
        "access_token": "string",
        "token_type": "string",
        "expires_in": "string"
      },
      "idToken": {
        "sub": "string",
        "email": "string",
        "email_verified": "string"
      },
      "userInfo": {
        "email": "string",
        "given_name": "string",
        "family_name": "string"
      },
      "samlResponse": {
        "string": "string"
      }
    }
  },
  "response": {
    "userAttributesToMap": {
      "string": "string"
    }
  }
}
```

インバウンドフェデレーションリクエストパラメータ

providerName

外部 ID プロバイダーの名前。

providerType

外部 ID プロバイダーのタイプ。有効な値:

OIDC、SAML、、Facebook、GoogleSignInWithApple、LoginWithAmazon。

属性

処理前に ID プロバイダーから受信した raw 属性。構造はプロバイダータイプによって異なります。

attributes.tokenResponse

/token エンドポイントからの OAuth トークンレスポンスデータ。OIDC およびソーシャルプロバイダーでのみ使用できます

す。access_token、id_token、、refresh_token、token_typeexpires_in、および が含まれますscope。

attributes.idToken

デコードおよび検証された ID トークン JWT クレーム。OIDC およびソーシャルプロバイダーでのみ使用できます。(一意のユーザー識別子)email、、sub、(iss発行者)name、aud (対象者)、exp (有効期限)、iat (発行時間) などの検証済みユーザー ID 情報が含まれます。

attributes.userInfo

UserInfo エンドポイントからの拡張ユーザープロフィール情報。OIDC およびソーシャルプロバイダーでのみ使用できます。given_name、、family_name、picture、などのプロバイダー固有のフィールドなどaddress、詳細なプロフィール属性が含まれます。IdP が UserInfo エンドポイントをサポートしていない場合、またはエンドポイント呼び出しが失敗した場合、空になる可能性があります。

attributes.samlResponse

SAML アサーション属性。SAML プロバイダーでのみ使用できます。SAML レスポンスの属性が含まれます。

インバウンドフェデレーションレスポンスパラメータ

userAttributesToMap

ユーザープロフィールに適用するユーザー属性。

⚠ Important

変更しない属性を含め、レスポンスに保持するすべてのユーザー属性を含める必要があります。userAttributesToMap レスポンスに含まれていない属性は削除され、ユーザープロフィールに保存されません。これは、変更された属性と変更されていない属性の両方に適用されます。

📘 空のレスポンス動作

に空のオブジェクトを返した場合 {}userAttributesToMap、ID プロバイダーからのすべての元の属性は変更されません。これは、Lambda 関数が実行されなかったかのように、no-op として機能します。これは、削除される属性を省略する場合とは異なります。

📘 プロバイダー固有の属性

の構造は、request.attributesによって異なりますproviderType。OIDC およびソーシャルプロバイダーには、tokenResponse、idToken、userInfo オブジェクトが含まれます。SAML プロバイダーには samlResponse オブジェクトのみが含まれます。

インバウンドフェデレーションの例: グループメンバーシップ管理

この例では、フェデレーティッド ID プロバイダーグループを Amazon Cognito ユーザープールグループにマッピングする方法を示します。この関数は、フェデレーティッドレスポンスからグループメンバーシップを抽出し、対応する Amazon Cognito グループにユーザーを自動的に追加するため、認証後のトリガーが不要になります。

Node.js

```
exports.handler = async (event) => {
  const { providerType, attributes } = event.request;

  // Extract user attributes based on provider type
  let userAttributesFromIdp = {};
  if (providerType === 'SAML') {
    userAttributesFromIdp = attributes.samlResponse || {};
  } else {
```

```
// For OIDC and Social providers, merge userInfo and idToken
userAttributesFromIdp = {
  ...(attributes.userInfo || {}),
  ...(attributes.idToken || {})
};
}

// Extract groups from federated response
const federatedGroups = userAttributesFromIdp.groups?.split(',') || [];

// Map federated groups to Cognito groups
const groupMapping = {
  'Domain Admins': 'Administrators',
  'Engineering': 'Developers',
  'Sales': 'SalesTeam'
};

// Filter to only in-scope groups
const mappedGroups = federatedGroups
  .map(group => groupMapping[group.trim()])
  .filter(group => group); // Remove undefined values

// Pass through attributes with mapped groups as custom attribute
const attributesToMap = {
  ...userAttributesFromIdp,
  'custom:user_groups': mappedGroups.join(',')
};

// Remove original groups attribute
delete attributesToMap.groups;

event.response.userAttributesToMap = attributesToMap;
return event;
};
```

Amazon Cognito は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

```
{
  "userPoolId": "us-east-1_XXXXXXXXX",
  "request": {
    "providerName": "CorporateAD",
    "providerType": "SAML",
    "attributes": {
      "samlResponse": {
        "email": "jane.smith@company.com",
        "given_name": "Jane",
        "family_name": "Smith",
        "groups": "Engineering,Domain Admins",
        "department": "Engineering"
      }
    }
  },
  "response": {
    "userAttributesToMap": {}
  }
}
```

インバウンドフェデレーションの例: 大きな属性を切り捨てる

この例では、Amazon Cognito のストレージ制限を超える属性値を切り捨てる方法を示します。この関数は、ID プロバイダーの各属性をチェックします。属性値が 2048 文字を超える場合、値は切り捨てられ、切り捨てを示す省略記号が追加されます。他のすべての属性は変更されずに渡されます。

Node.js

```
exports.handler = async (event) => {
  const MAX_ATTRIBUTE_LENGTH = 2048;

  // Get the identity provider attributes based on provider type
  const { providerType, attributes } = event.request;
  let idpAttributes = {};

  if (providerType === 'SAML') {
    idpAttributes = attributes.samlResponse || {};
  } else {
    // For OIDC and Social providers, merge userInfo and idToken
    idpAttributes = {
```

```
        ...(attributes.userInfo || {}),
        ...(attributes.idToken || {})
    };
}

const userAttributes = {};

// Process each attribute
for (const [key, value] of Object.entries(idpAttributes)) {
    if (typeof value === 'string' && value.length > MAX_ATTRIBUTE_LENGTH) {
        // Truncate the value and add ellipsis
        userAttributes[key] = value.substring(0, MAX_ATTRIBUTE_LENGTH - 3) +
        '...';
        console.log(`Truncated attribute ${key} from ${value.length} to
        ${userAttributes[key].length} characters`);
    } else {
        // Keep the original value
        userAttributes[key] = value;
    }
}

// Return the modified attributes
event.response.userAttributesToMap = userAttributes;
return event;
};
```

Amazon Cognito は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

```
{
  "version": "string",
  "triggerSource": "InboundFederation_ExternalProvider",
  "region": "us-east-1",
  "userPoolId": "us-east-1_XXXXXXXXXX",
  "userName": "ExampleProvider_12345",
  "callerContext": {
    "awsSdkVersion": "string",
    "clientId": "string"
```

```

    },
    "request": {
      "providerName": "ExampleProvider",
      "providerType": "OIDC",
      "attributes": {
        "tokenResponse": {
          "access_token": "abcDE...",
          "token_type": "Bearer",
          "expires_in": "3600"
        },
        "idToken": {
          "sub": "12345",
          "email": "user@example.com"
        },
        "userInfo": {
          "email": "user@example.com",
          "given_name": "Example",
          "family_name": "User",
          "bio": "This is a very long biography that contains more than 2048
characters..."
        }
      }
    },
    "response": {
      "userAttributesToMap": {}
    }
  }
}

```

インバウンドフェデレーションの例: フェデレーションイベントのログ記録

この例では、モニタリングとデバッグのためにフェデレーティッド認証イベントを記録する方法を示します。このサンプル関数は、フェデレーティッドユーザーとその属性に関する詳細情報をキャプチャし、認証プロセスを可視化します。

Node.js

```

exports.handler = async (event) => {
  const { providerName, providerType, attributes } = event.request;

  // Extract user attributes based on provider type
  let userAttributesFromIdp = {};
  if (providerType === 'SAML') {

```



```

        userAttributesFromIdp = attributes.samlResponse || {};
    } else {
        // For OIDC and Social providers, merge userInfo and idToken
        userAttributesFromIdp = {
            ...(attributes.userInfo || {}),
            ...(attributes.idToken || {})
        };
    }

    // Log federated authentication details
    console.log(JSON.stringify({
        timestamp: new Date().toISOString(),
        providerName,
        providerType,
        userEmail: userAttributesFromIdp.email,
        attributeCount: Object.keys(userAttributesFromIdp).length,
        attributes: userAttributesFromIdp
    }));

    // Pass through all attributes unchanged
    event.response.userAttributesToMap = userAttributesFromIdp;
    return event;
};

```

Amazon Cognito は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

```

{
  "version": "string",
  "triggerSource": "InboundFederation_ExternalProvider",
  "region": "us-east-1",
  "userPoolId": "us-east-1_XXXXXXXXXX",
  "userName": "CorporateAD_john.doe",
  "callerContext": {
    "awsSdkVersion": "string",
    "clientId": "string"
  },
  "request": {

```

```
"providerName": "CorporateAD",
"providerType": "SAML",
"attributes": {
  "samlResponse": {
    "email": "john.doe@company.com",
    "given_name": "John",
    "family_name": "Doe",
    "department": "Engineering",
    "employee_id": "EMP12345"
  }
},
"response": {
  "userAttributesToMap": {}
}
}
```

予想される CloudWatch Logs 出力:

JSON

```
{
  "timestamp": "2025-01-14T21:17:40.153Z",
  "providerName": "CorporateAD",
  "providerType": "SAML",
  "userEmail": "john.doe@company.com",
  "attributeCount": 5,
  "attributes": {
    "email": "john.doe@company.com",
    "given_name": "John",
    "family_name": "Doe",
    "department": "Engineering",
    "employee_id": "EMP12345"
  }
}
```

カスタム認証チャレンジの Lambda トリガー

Amazon Cognito ユーザープールの認証フローを構築した後で、組み込みフローを超えた認証モデルに拡張したい場合があります。カスタムチャレンジトリガーの一般的なユースケースの 1 つは、

ユーザー名、パスワード、および多要素認証 (MFA) を超えた追加のセキュリティチェックを実装することです。カスタムチャレンジは、Lambda がサポートするプログラミング言語で生成できる質問とレスポンスです。例えば、認証を許可する前に、CAPTCHA を解決することや、セキュリティの質問に回答することをユーザーに要求する場合があります。もう 1 つの潜在的なニーズは、特殊な認証要素やデバイスと統合することです。または、ハードウェアセキュリティキーまたは生体認証デバイスでユーザーを認証するソフトウェアを開発済みである場合もあります。カスタムチャレンジの認証成功の定義は、どのような回答であっても、Lambda 関数が正しいものとして受け入れる回答です。例えば、固定文字列、または外部 API からの十分なレスポンスです。

カスタムチャレンジを使用した認証を開始して認証プロセスを完全に制御することも、アプリケーションがカスタムチャレンジを受信する前にユーザー名パスワード認証を実行することもできます。

カスタム認証チャレンジの Lambda トリガー

定義

チャレンジシーケンスを開始します。新しいチャレンジを開始するか、認証を完了としてマークするか、認証の試行を停止するかを決定します。

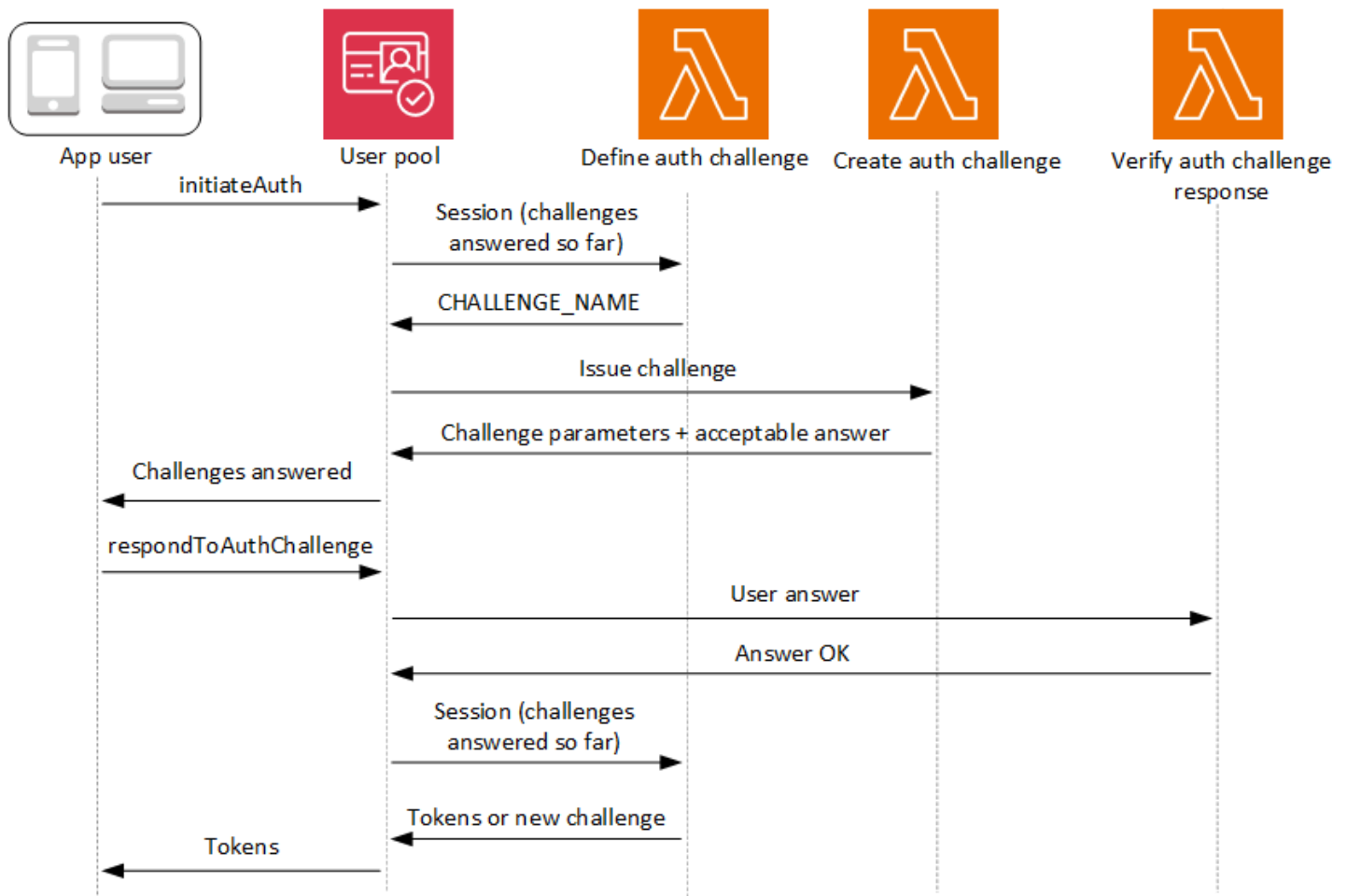
作成

ユーザーが回答する必要がある質問をアプリケーションに発行します。この関数は、アプリケーションがユーザーに表示するセキュリティの質問や CAPTCHA へのリンクを提示する場合があります。

検証

予想される回答を把握し、それをチャレンジレスポンスでアプリケーションが提供する回答と比較します。この関数は、CAPTCHA サービスの API を呼び出して、ユーザーの試行したソリューションの期待される結果を取得することができます。

これらの 3 つの Lambda 関数は連鎖して、完全に制御範囲内で独自の設計の認証メカニズムを提示します。カスタム認証は、クライアントと Lambda 関数にアプリケーションロジックを必要とするため、マネージドログイン内ではカスタム認証を処理できません。この認証システムには、デベロッパーの労力が追加が必要です。アプリケーションは、ユーザープール API を使用して認証フローを実行し、結果として返されるチャレンジを、カスタム構築されたログインインターフェイスで処理する必要があります。このインターフェイスは、カスタム認証チャレンジの中心に質問を表示します。



カスタム認証の実装の詳細については、「[カスタム認証フローとチャレンジ](#)」を参照してください。

API オペレーション [InitiateAuth](#) または [AdminInitiateAuth](#) と [RespondToAuthChallenge](#) または [AdminRespondToAuthChallenge](#) 間の認証。このフローでは、認証が失敗するか、トークンが発行されるまで、ユーザーは引き続きチャレンジに回答して認証を行います。チャレンジレスポンスが新しいチャレンジとなる場合があります。この場合、アプリケーションは新しいチャレンジに対して必要な回数を応答します。認証は、認証チャレンジの定義関数がそれまでの結果を分析し、すべてのチャレンジが回答されたと判断し、`IssueTokens` を返すときに完了します。

トピック

- [カスタムチャレンジフローでの SRP 認証](#)
- [認証チャレンジの定義の Lambda トリガー](#)
- [認証チャレンジの作成の Lambda トリガー](#)
- [認証チャレンジレスポンスの検証の Lambda トリガー](#)

カスタムチャレンジフローでの SRP 認証

Amazon Cognito では、カスタムチャレンジを発行する前にユーザーパスワードを検証させることができます。[リクエストトレートクォータ](#)の認証カテゴリに関連付けられているすべての Lambda トリガーは、カスタムチャレンジフローで SRP 認証を行うと、実行されます。以下は、そのプロセスの概要です。

1. アプリは、AuthParameters マップを使用して InitiateAuth または AdminInitiateAuth を呼び出してサインインを開始します。パラメータには CHALLENGE_NAME: SRP_A、SRP_A および USERNAME の値を含める必要があります。
2. Amazon Cognito は、challengeName: SRP_A と challengeResult: true を含む初期セッションで、認証チャレンジの定義 Lambda トリガーを呼び出します。
3. Lambda 関数は、これらの入力を受け取った後、challengeName: PASSWORD_VERIFIER、issueTokens: false、failAuthentication: false で応答します。
4. パスワードの検証が成功すると、Amazon Cognito challengeName: PASSWORD_VERIFIER と challengeResult: true が含まれる新しいセッションで Lambda 関数を再度呼び出します。
5. カスタムチャレンジを開始するために、Lambda 関数は challengeName: CUSTOM_CHALLENGE、issueTokens: false、および failAuthentication: false で応答します。パスワード検証でカスタム認証フローを開始したくない場合は、CHALLENGE_NAME: CUSTOM_CHALLENGE を含む AuthParameters マップでサインインを開始できます。
6. チャレンジループは、すべてのチャレンジが回答されるまで繰り返します。

以下は、SRP フローによるカスタム認証の前になされる開始 InitiateAuth リクエストの例です。

```
{
  "AuthFlow": "CUSTOM_AUTH",
  "ClientId": "1example23456789",
  "AuthParameters": {
    "CHALLENGE_NAME": "SRP_A",
    "USERNAME": "testuser",
    "SRP_A": "[SRP_A]",
    "SECRET_HASH": "[secret hash]"
  }
}
```

カスタム認証 SRP フローでのパスワードのリセット

ユーザーが `FORCE_CHANGE_PASSWORD` ステータスの場合、カスタム認証フローは、認証チャレンジの整合性を維持しながら、パスワード変更ステップを統合する必要があります。Amazon Cognito は、`NEW_PASSWORD_REQUIRED` チャレンジ中に [\[認証チャレンジを定義\]](#) Lambda トリガーを呼び出します。このシナリオの場合、ユーザーがカスタムチャレンジフローと SRP 認証を使用してサインインするときに、パスワードのリセット状態にあれば、新しいパスワードを設定できます。

ユーザーは、`RESET_REQUIRED` ステータスまたは `FORCE_CHANGE_PASSWORD` ステータスである場合、`NEW_PASSWORD` を使用して `NEW_PASSWORD_REQUIRED` チャレンジに [応答](#) する必要があります。SRP を使用したカスタム認証の場合、ユーザーが SRP `NEW_PASSWORD_REQUIRED` チャレンジを完了すると、Amazon Cognito は `PASSWORD_VERIFIER` チャレンジを返します。認証チャレンジ定義トリガーは、両方のチャレンジ結果を `session` 配列で受け取り、ユーザーがパスワードを正常に変更した後で、追加のカスタムチャレンジを続行できます。

認証チャレンジ定義の Lambda トリガーは、SRP 認証、パスワードリセット、以降のカスタムチャレンジを通じて、チャレンジシーケンスを管理する必要があります。トリガーは、完了したチャレンジの配列 (`PASSWORD_VERIFIER` と `NEW_PASSWORD_REQUIRED` の両方の結果を含む) を `session` パラメータで受け取ります。実装の例については、「[認証チャレンジの定義の例](#)」を参照してください。

認証フローの手順

ユーザーがカスタムチャレンジの前にパスワードを検証する必要がある場合は、次の手順に従います。

1. アプリは、`AuthParameters` マップを使用して `InitiateAuth` または `AdminInitiateAuth` を呼び出してサインインを開始します。パラメータには `CHALLENGE_NAME`: `SRP_A` に加えて、`SRP_A` と `USERNAME` の値を含める必要があります。
2. Amazon Cognito は、`challengeName`: `SRP_A` と `challengeResult`: `true` を含む初期セッションで、認証チャレンジの定義 Lambda トリガーを呼び出します。
3. Lambda 関数は、これらの入力を受け取った後、`challengeName`: `PASSWORD_VERIFIER`、`issueTokens`: `false`、`failAuthentication`: `false` で応答します。
4. パスワードの検証が成功すると、次の 2 つのどちらかになります。

通常ステータスのユーザーの場合:

Amazon Cognito は、`challengeName`: `PASSWORD_VERIFIER` と `challengeResult`: `true` を含む新しいセッションで Lambda 関数を再度呼び出します。

カスタムチャレンジを開始するために、Lambda 関数は `challengeName`: `CUSTOM_CHALLENGE`、`issueTokens`: `false`、および `failAuthentication`: `false` で応答します。

RESET_REQUIRED ステータスまたは **FORCE_CHANGE_PASSWORD** ステータスのユーザーの場合:

Amazon Cognito は、`challengeName`: `PASSWORD_VERIFIER` と `challengeResult`: `true` を含むセッションで Lambda 関数を呼び出します。

Lambda 関数は、`challengeName`: `NEW_PASSWORD_REQUIRED`、`issueTokens`: `false`、および `failAuthentication`: `false` により、これに応答する必要があります。

パスワードが正常に変更されると、Amazon Cognito は `PASSWORD_VERIFIER` と `NEW_PASSWORD_REQUIRED` の両方の結果を含むセッションで Lambda 関数を呼び出します。

カスタムチャレンジを開始するために、Lambda 関数は `challengeName`: `CUSTOM_CHALLENGE`、`issueTokens`: `false`、および `failAuthentication`: `false` で応答します。

5. チャレンジループは、すべてのチャレンジが回答されるまで繰り返します。

パスワード検証でカスタム認証フローを開始したくない場合は、`CHALLENGE_NAME`: `CUSTOM_CHALLENGE` を含む `AuthParameters` マップでサインインを開始できます。

セッション管理

認証フローは、一連のセッション ID とチャレンジ結果を通じてセッションの継続性を維持します。チャレンジレスポンスごとに新しいセッション ID を生成して、セッション再利用エラーを防ぎます。これは、多要素認証フローでは特に重要です。

チャレンジの結果は、Lambda トリガーが受信するセッション配列に時系列で保存されます。 **FORCE_CHANGE_PASSWORD** ステータスのユーザーの場合、セッション配列には以下が含まれます。

1. `session[0]` - 最初の `SRP_A` チャレンジ
2. `session[1]` - `PASSWORD_VERIFIER` の結果
3. `session[2]` - `NEW_PASSWORD_REQUIRED` の結果
4. 後続の要素 - 追加のカスタムチャレンジの結果

認証フローの例

次の例は、パスワード変更とカスタム CAPTCHA チャレンジの両方を完了する必要があるユーザー (FORCE_CHANGE_PASSWORD ステータス) の完全なカスタム認証フローを示しています。

1. InitiateAuth リクエスト

```
{
  "AuthFlow": "CUSTOM_AUTH",
  "ClientId": "1example23456789",
  "AuthParameters": {
    "CHALLENGE_NAME": "SRP_A",
    "USERNAME": "testuser",
    "SRP_A": "[SRP_A]"
  }
}
```

2. InitiateAuth レスポンス

```
{
  "ChallengeName": "PASSWORD_VERIFIER",
  "ChallengeParameters": {
    "USER_ID_FOR_SRP": "testuser"
  },
  "Session": "[session_id_1]"
}
```

3. PASSWORD_VERIFIER を使用した RespondToAuthChallenge リクエスト

```
{
  "ChallengeName": "PASSWORD_VERIFIER",
  "ClientId": "1example23456789",
  "ChallengeResponses": {
    "PASSWORD_CLAIM_SIGNATURE": "[claim_signature]",
    "PASSWORD_CLAIM_SECRET_BLOCK": "[secret_block]",
    "TIMESTAMP": "[timestamp]",
    "USERNAME": "testuser"
  },
  "Session": "[session_id_1]"
}
```

4. NEW_PASSWORD_REQUIRED チャレンジを使用した RespondToAuthChallenge レスポンス


```
{
  "ChallengeName": "NEW_PASSWORD_REQUIRED",
  "ChallengeParameters": {},
  "Session": "[session_id_2]"
}
```

5. NEW_PASSWORD_REQUIRED を使用した RespondToAuthChallenge リクエスト

```
{
  "ChallengeName": "NEW_PASSWORD_REQUIRED",
  "ClientId": "1example23456789",
  "ChallengeResponses": {
    "NEW_PASSWORD": "[password]",
    "USERNAME": "testuser"
  },
  "Session": "[session_id_2]"
}
```

6. CAPTCHA カスタムチャレンジを使用した RespondToAuthChallenge レスポンス

```
{
  "ChallengeName": "CUSTOM_CHALLENGE",
  "ChallengeParameters": {
    "captchaUrl": "url/123.jpg"
  },
  "Session": "[session_id_3]"
}
```

7. CAPTCHA カスタムチャレンジを使用した RespondToAuthChallenge リクエスト

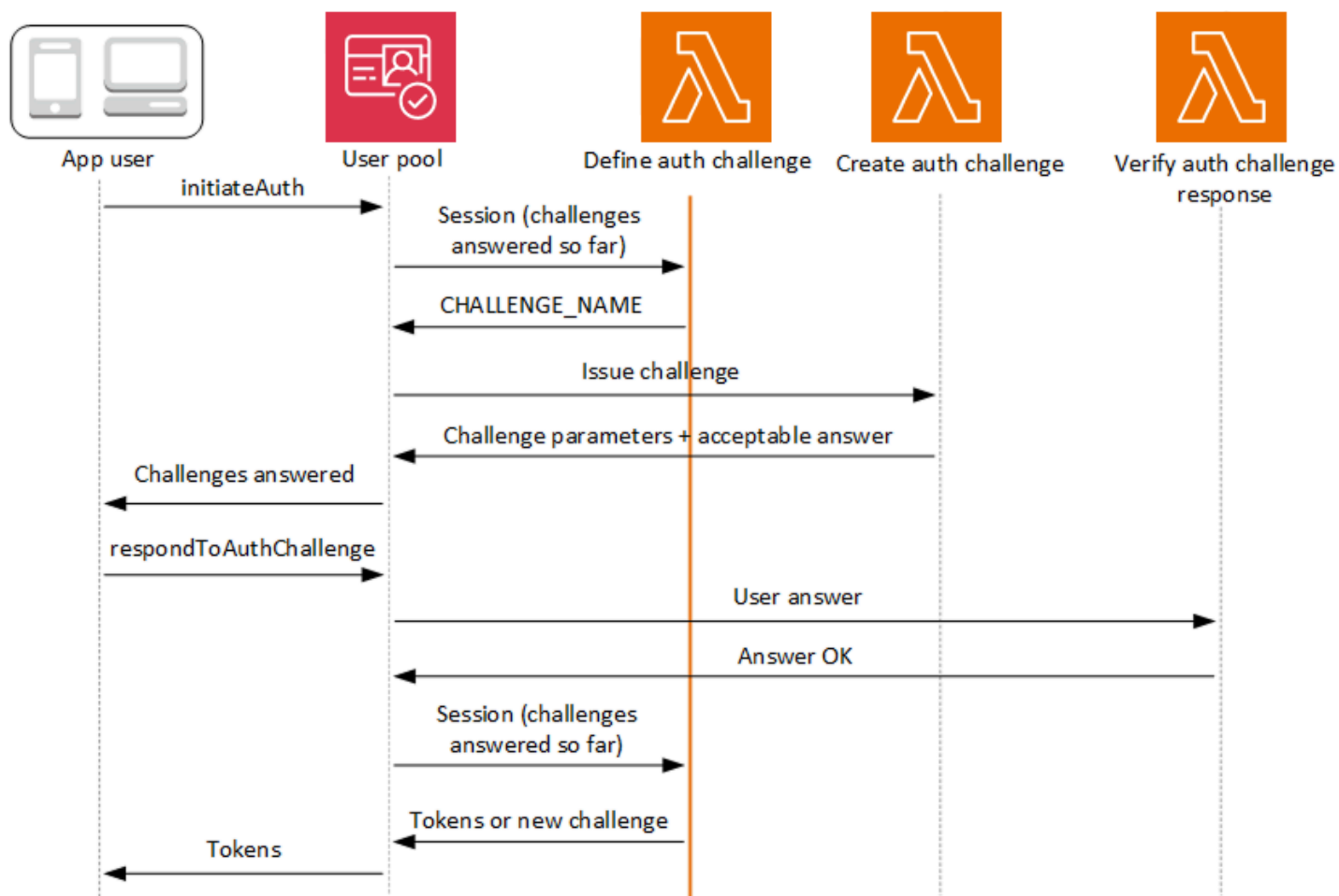
```
{
  "ChallengeName": "CUSTOM_CHALLENGE",
  "ClientId": "1example23456789",
  "ChallengeResponses": {
    "ANSWER": "123",
    "USERNAME": "testuser"
  },
  "Session": "[session_id_3]"
}
```

6. 最終的な成功レスポンス

```
{
  "AuthenticationResult": {
    "AccessToken": "eyJra456defEXAMPLE",
    "ExpiresIn": 3600,
    "IdToken": "eyJra789ghiEXAMPLE",
    "RefreshToken": "eyJjd123abcEXAMPLE",
    "TokenType": "Bearer"
  },
  "ChallengeParameters": {}
}
```

認証チャレンジの定義の Lambda トリガー

認証チャレンジ定義トリガーは、カスタム認証フローでチャレンジシーケンスを維持する Lambda 関数です。チャレンジシーケンスの成功または失敗を宣言し、シーケンスがまだ完了していない場合に次のチャレンジを設定します。



認証チャレンジの定義

Amazon Cognito は、このトリガーを呼び出して[カスタム認証フロー](#)を開始します。

この Lambda トリガーのリクエストには、`session` が含まれます。`session` パラメータは、現在の認証プロセスでユーザーに提示されたすべてのチャレンジが含まれる配列です。リクエストには、対応する結果も含まれます。`session` 配列は、チャレンジの詳細 (`ChallengeResult`) を時系列に保存します。チャレンジ `session[0]` は、ユーザーが最初に受け取るチャレンジを表します。

トピック

- [認証チャレンジの定義の Lambda トリガーのパラメータ](#)
- [認証チャレンジの定義の例](#)

認証チャレンジの定義の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

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

```
}  
}
```

認証チャレンジの定義のリクエストパラメータ

Amazon Cognito が Lambda 関数を呼び出すときに、次のパラメータを提供します。

userAttributes

ユーザー属性を表す 1 つ以上の名前 - 値ペア。

userNotFound

ユーザープールクライアントの `PreventUserExistenceErrors` が `ENABLED` に設定されている場合に、Amazon Cognito が入力するブール値です。true の値は、ユーザー ID (ユーザー名、メールアドレス、その他の詳細など) が既存のいずれのユーザーとも一致しなかったことを意味します。`PreventUserExistenceErrors` が `ENABLED` に設定されている場合、サービスは存在しないユーザーをアプリに通知しません。Lambda 関数が同じユーザーエクスペリエンスを維持し、レイテンシーを考慮することをお勧めします。この方法では、発信者はユーザーが存在する場合と存在しない場合で異なる動作を検出することができません。

セッション

`ChallengeResult` 要素の配列。それぞれに以下の要素が含まれます。

challengeName

次のチャレンジタイプのいずれかです:

`CUSTOM_CHALLENGE`、`SRP_A`、`PASSWORD_VERIFIER`、`SMS_MFA`、`EMAIL_OTP`、`SOFTWARE_TOKEN_MFA` または `ADMIN_NO_SRP_AUTH`。

認証チャレンジの定義機能が、多要素認証を設定したユーザーに `PASSWORD_VERIFIER` チャレンジを発行すると、Amazon Cognito はそれに続いて `SMS_MFA`、`EMAIL_OTP`、または `SOFTWARE_TOKEN_MFA` のチャレンジを行います。これらは、多要素認証コードのプロンプトです。関数には、`SMS_MFA`、`EMAIL_OTP`、および `SOFTWARE_TOKEN_MFA` のチャレンジからの入カイベントの処理を含めます。認証チャレンジの定義関数から MFA チャレンジを呼び出す必要はありません。

⚠ Important

ユーザーが正常に認証され、トークンが発行されるべきかを判断するときは、常に `define auth challenge` 関数で `challengeName` をチェックし、予想された値と一致することを確認します。

challengeResult

ユーザーが正常にチャレンジを完了した場合は `true` に、それ以外の場合は `false` に設定します。

challengeMetadata

カスタムチャレンジの名前を入力します。challengeName が `CUSTOM_CHALLENGE` である場合にのみ使用されます。

clientMetadata

認証チャレンジの定義のトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。このデータを Lambda 関数に渡すには、[AdminRespondToAuthChallenge](#) および [RespondToAuthChallenge](#) API オペレーションで `ClientMetadata` パラメータを使用できます。define auth challenge 関数を呼び出すリクエストに、[AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションの `ClientMetadata` パラメータに渡されたデータは含まれません。

認証チャレンジの定義のレスポンスパラメータ

レスポンスで、認証プロセスの次のステージを返すことができます。

challengeName

次のチャレンジの名前を含む文字列。ユーザーに新しいチャレンジを提示する場合は、ここでチャレンジ名を指定します。

issueTokens

ユーザーが認証チャレンジを十分に完了したと判断した場合、`true` に設定します。ユーザーがチャレンジを十分に満たしていない場合は、`false` に設定します。

failAuthentication

現在の認証プロセスを終了する場合は、`true` に設定します。現在の認証プロセスを続行するには、`false` に設定します。

認証チャレンジの定義の例

この例では、認証用に一連のチャレンジを定義し、それらのチャレンジがすべて正常に完了した場合にのみトークンを発行します。ユーザーが `SRP_A` チャレンジと `PASSWORD_VERIFIER` チャレンジを使用して SRP 認証を完了すると、この関数は、認証チャレンジの作成トリガーを呼び出す `CUSTOM_CHALLENGE` を、ユーザーに渡します。[認証チャレンジの作成例](#)と組み合わせることで、このシーケンスは、チャレンジ 3 で CAPTCHA チャレンジを提供し、チャレンジ 4 でセキュリティの質問を提供します。

ユーザーが CAPTCHA を解決してセキュリティの質問に回答すると、この関数は、ユーザープールがトークンを発行できることを確認します。SRP 認証は必須ではありません。CAPTCHA とセキュリティの質問をチャレンジ 1 とチャレンジ 2 として設定することもできます。認証チャレンジの定義関数が SRP チャレンジを宣言しない場合、ユーザーの成功はカスタムプロンプトへの応答によってのみ決定されます。

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" &&
```

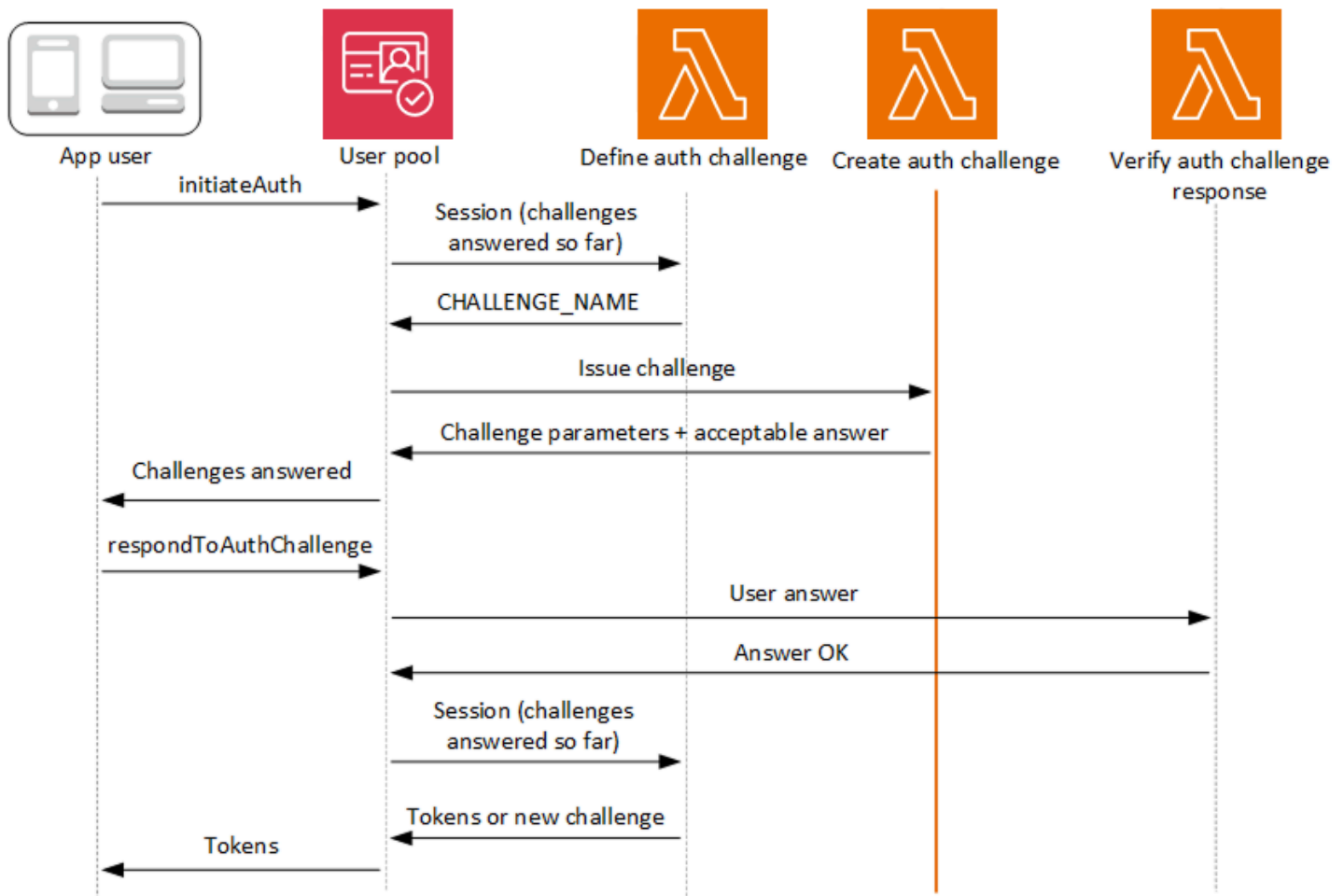
```
    event.request.session[2].challengeResult === true
  ) {
    event.response.issueTokens = false;
    event.response.failAuthentication = false;
    event.response.challengeName = "CUSTOM_CHALLENGE";
  } else if (
    event.request.session.length === 4 &&
    event.request.session[3].challengeName === "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 };
```

認証チャレンジの作成の Lambda トリガー

認証チャレンジの作成のトリガーは、認証チャレンジの定義トリガーによって宣言された各チャレンジの詳細を持つ Lambda 関数です。これは、認証チャレンジの定義トリガーによって宣言されたチャレンジ名を処理し、アプリケーションがユーザーに提示する必要がある `publicChallengeParameters` を返します。次に、この関数は、ユーザープールが認証チャレンジ検証トリガーに渡すチャレンジに対する回答、`privateChallengeParameters` をユーザープールに提供します。認証チャレンジの定義トリガーがチャレンジシーケンスを管理する場合、認証チャレンジの作成トリガーはチャレンジの内容を管理します。



認証チャレンジの作成

認証チャレンジの定義トリガーの一部としてカスタムチャレンジが指定されている場合、Amazon Cognito は認証チャレンジの定義後にこのトリガーを呼び出します。これにより、[カスタム認証フロー](#)を作成します。

この Lambda トリガーは、ユーザーに提示するチャレンジを作成するために呼び出されます。この Lambda トリガーのリクエストには `challengeName` と `session` が含まれます。`challengeName` は文字列であり、ユーザーに対する次のチャレンジの名前です。この属性の値は、認証チャレンジの定義の Lambda トリガーに設定されています。

チャレンジループは、すべてのチャレンジに応答するまで繰り返されます。

トピック

- [認証チャレンジの作成 Lambda トリガーパラメータ](#)
- [認証チャレンジの作成の例](#)

認証チャレンジの作成 Lambda トリガーパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する [共通パラメータ](#) を組み合わせたものです。

JSON

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

認証チャレンジの作成のリクエストパラメータ

userAttributes

ユーザー属性を表す 1 つ以上の名前 - 値ペア。

userNotFound

このブール値は、ユーザープールクライアントで `PreventUserExistenceErrors` が `ENABLED` に設定されている場合に設定されます。

challengeName

新しいチャレンジの名前。

session

`session` 要素は `ChallengeResult` 要素の配列であり、それぞれに以下の要素が含まれます。

challengeName

チャレンジタイ

プ。"CUSTOM_CHALLENGE"、"PASSWORD_VERIFIER"、"SMS_MFA"、"DEVICE_SRP_AUTH"、"D" または "ADMIN_NO_SRP_AUTH" のいずれか。

challengeResult

ユーザーが正常にチャレンジを完了した場合は `true` に、それ以外の場合は `false` に設定します。

challengeMetadata

カスタムチャレンジの名前を入力します。challengeName が "CUSTOM_CHALLENGE" である場合にのみ使用されます。

clientMetadata

認証チャレンジの作成のトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。このデータは、[AdminRespondToAuthChallenge](#) および [RespondToAuthChallenge](#) API アクションで `ClientMetadata` パラメータを使用することによって Lambda 関数に渡すことができます。create auth challenge 関数を呼び出すリクエストに、[AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションで `ClientMetadata` パラメータに渡されたデータは含まれません。

認証チャレンジの作成のレスポンスパラメータ

publicChallengeParameters

クライアントアプリケーションでユーザーに提示されるチャレンジに使用する 1 つ以上のキー - 値ペア。このパラメータには、ユーザーにチャレンジを正確に提示するために必要なすべての情報を含める必要があります。

privateChallengeParameters

このパラメータは、認証チャレンジレスポンスの検証の Lambda トリガー以外では使用されません。このパラメータには、チャレンジに対するユーザーのレスポンスを検証するために必要な情報のすべてを含める必要があります。つまり、publicChallengeParameters には、ユーザーに示される質問が含まれ、privateChallengeParameters には、質問に対する有効な回答が含まれます。

challengeMetadata

カスタムチャレンジの名前 (カスタムチャレンジの場合)。

認証チャレンジの作成の例

この関数には、[認証チャレンジの定義例](#)のチャレンジシーケンスに対応する 2 つのカスタムチャレンジがあります。最初の 2 つのチャレンジは SRP 認証です。3 番目のチャレンジの場合、この関数はチャレンジレスポンスで CAPTCHA URL をアプリケーションに返します。アプリケーションは、指定された URL で CAPTCHA をレンダリングし、ユーザーの入力を返します。CAPTCHA イメージの URL が "captchaUrl" としてパブリックチャレンジパラメータに追加され、想定される回答がプライベートチャレンジパラメータに追加されます。

4 番目のチャレンジの場合、この関数はセキュリティの質問を返します。アプリケーションは質問をレンダリングし、ユーザーに回答を求めます。ユーザーが両方のカスタムチャレンジを解決すると、認証チャレンジの定義トリガーは、ユーザープールがトークンを発行できることを確認します。

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";
}

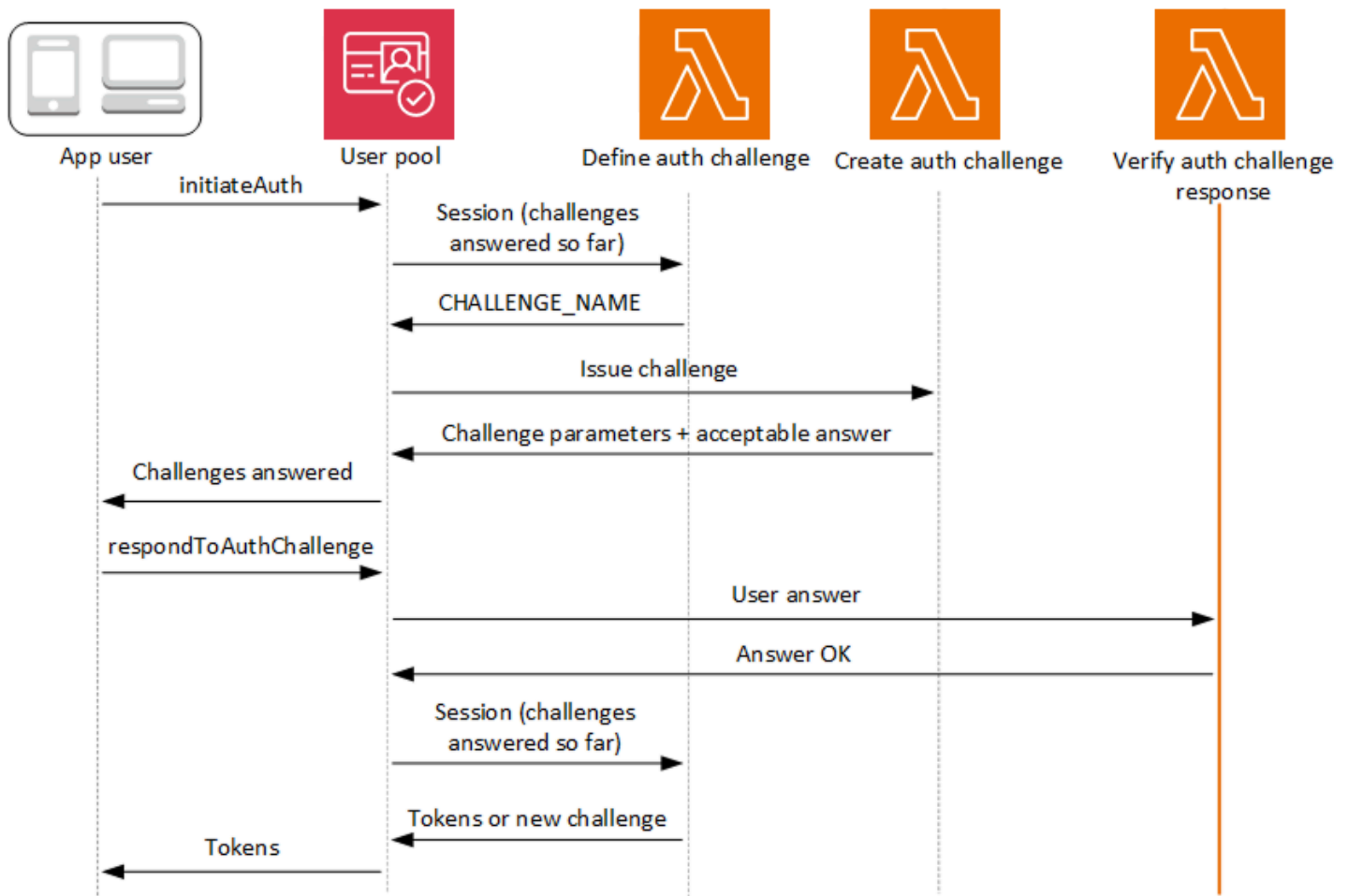
return event;
};

export { handler };

```

認証チャレンジレスポンスの検証の Lambda トリガー

認証チャレンジの検証トリガーは、ユーザーが提供したレスポンスを既知のレスポンスと比較する Lambda 関数です。この関数は、ユーザーがチャレンジに正しく応答したかどうかをユーザープールに伝えます。認証チャレンジの検証トリガーが true の `answerCorrect` で応答すると、認証シーケンスは続行できます。



認証チャレンジレスポンスの検証

Amazon Cognito は、このトリガーが呼び出して、カスタム認証チャレンジに対するユーザーからのレスポンスが有効であるかどうかを検証します。これはユーザープールの[カスタム認証フロー](#)の一環です。

このトリガーのリクエストには `privateChallengeParameters` および `challengeAnswer` パラメータが含まれます。`privateChallengeParameters` 値は、認証チャレンジの作成の Lambda トリガーによって返され、ユーザーからの期待されるレスポンスが含まれます。`challengeAnswer` パラメータには、チャレンジに対するユーザーのレスポンスが含まれます。

レスポンスには、`answerCorrect` 属性が含まれます。ユーザーがチャレンジを正常に完了すると、Amazon Cognito は属性値を `true` に設定します。ユーザーがチャレンジを正常に完了しなかった場合、Amazon Cognito は値を `false` に設定します。

チャレンジループは、すべてのチャレンジが回答されるまで繰り返します。

トピック

- [認証チャレンジの検証の Lambda トリガーのパラメータ](#)
- [認証チャレンジレスポンスの確認の例](#)

認証チャレンジの検証の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

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

```
        "string": "string",
        . . .
    },
    "userNotFound": boolean
},
"response": {
    "answerCorrect": boolean
}
}
```

認証チャレンジの検証のリクエストパラメータ

userAttributes

このパラメータには、ユーザー属性を表す 1 つ以上の名前-値ペアが含まれます。

userNotFound

Amazon Cognito がユーザープールクライアントの `PreventUserExistenceErrors` を `ENABLED` に設定すると、Amazon Cognito はこのブール値を入力します。

privateChallengeParameters

このパラメータは、認証チャレンジの作成トリガーから取得されます。ユーザーがチャレンジに合格したかどうかを判断するために、Amazon Cognito はパラメータをユーザーの `challengeAnswer` パラメータと比較します。

このパラメータには、チャレンジに対するユーザーのレスポンスを検証するために必要な情報のすべてが含まれます。その情報には、Amazon Cognito がユーザーに提示する質問 (`publicChallengeParameters`) と、その質問に対する有効な回答 (`privateChallengeParameters`) が含まれます。認証チャレンジレスポンスの検証の Lambda トリガーのみがこのパラメータを使用します。

challengeAnswer

このパラメータ値は、チャレンジに対するユーザーの回答です。

clientMetadata

このパラメータには、認証チャレンジの検証のトリガーの Lambda 関数へのカスタム入力として提供できる 1 つまたは複数のキーバリューペアが含まれています。このデータを Lambda 関数に渡すには、[AdminRespondToAuthChallenge](#) および [RespondToAuthChallenge](#) API オペレーションで `ClientMetadata` パラメータを使用します。Amazon Cognito は、認証チャレンジの検証関数

に渡すリクエストの [AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションの ClientMetadata パラメータからのデータを含めません。

認証チャレンジの検証のレスポンスパラメータ

answerCorrect

ユーザーがチャレンジを正常に完了した場合、Amazon Cognito はこのパラメータを true に設定します。ユーザーがチャレンジを正常に完了しなかった場合、Amazon Cognito はパラメータを false に設定します。

認証チャレンジレスポンスの確認の例

この認証チャレンジの確認関数は、チャレンジに対するユーザーのレスポンスが、想定されたレスポンスと一致するかどうかを確認します。ユーザーの回答はアプリケーションからの入力によって定義され、推奨される回答は [認証チャレンジの作成トリガーレスポンス](#) からのレスポンスの privateChallengeParameters.answer によって定義されます。正しい回答と与えられた回答は、どちらもこの関数への入力イベントの一部となります。

この例では、ユーザーのレスポンスが想定されたレスポンスと一致した場合、Amazon Cognito は answerCorrect パラメータを true に設定します。

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 };
```

トークン生成前の Lambda トリガー

Amazon Cognito は、このトリガーをトークンの生成前に呼び出すため、ユーザープールトークンのクレームをカスタマイズできます。バージョン 1 または V1_0 のトークン生成前トリガーイベントの基本機能を使用して、アイデンティティ (ID) トークンをカスタマイズできます。エッセンシャル機能プランまたはプラス機能プランを使用するユーザープールでは、Machine to Machine (M2M) クライアント認証情報の付与で、アクセストークンのカスタマイズを伴うバージョン 2 (V2_0) トリガーイベントと、アクセストークンのカスタマイズを伴うバージョン 3 (V3_0) トリガーイベントを生成できます。

Amazon Cognito は、ID トークンに書き込むデータが含まれているリクエストとして V1_0 イベントを関数に送信します。V2_0 イベントまたは V3_0 イベントは、Amazon Cognito が ID トークンとアクセストークンの両方に書き込むデータが含まれている単一のリクエストです。両方のトークンをカスタマイズするには、トリガーバージョン 2 または 3 を使用するように関数を更新し、両方のトークンのデータを同じレスポンスで送信する必要があります。

Amazon Cognito は、バージョン 2 のイベントレスポンスを、ユーザー認証 (ユーザー自らユーザープールに提示した認証情報) からのアクセストークンに適用します。バージョン 3 のイベントレスポンスは、ユーザー認証とマシン認証からのアクセストークンに適用されます。マシン認証では、自動化されたシステムが、アプリケーションクライアントシークレットを使用してアクセストークンリクエストを承認します。結果のアクセストークンの状況を除いて、バージョン 2 と 3 のイベントは同じです。

この Lambda トリガーでは、Amazon Cognito がアプリケーションに発行する前の ID トークンとアクセストークンの一部のクレームを追加、削除、変更できます。この機能を使用するには、Amazon Cognito ユーザープールコンソールから Lambda 関数を関連付けるか、AWS Command Line Interface (AWS CLI) でユーザープール LambdaConfig を更新します。

イベントバージョン

ユーザープールは、トークン生成前トリガーイベントのさまざまなバージョンを Lambda 関数に配信できます。V1_0 トリガーは、ID トークンを変更するためのパラメータを配信します。V2_0 または V3_0 トリガーは、以下のパラメータを提供します。

1. V1_0 トリガーの関数。
2. アクセストークンをカスタマイズする機能。
3. 複雑なデータ型を ID に渡し、次のトークンクレーム値にアクセスする機能。
 - String

- Number
- ブール値
- 文字列、数値、ブール値、またはこれらの組み合わせの配列
- JSON

Note

ID トークンでは、`phone_number_verified`、`email_verified`、`updated_at` および `address` を除くクレームの値に複雑なオブジェクトを入力できます。

ユーザープールはデフォルトで V1_0 イベントを配信します。V2_0 イベントを送信するようにユーザープールを設定するには、Amazon Cognito コンソールでトリガーを設定するときに、[トリガーイベントバージョン] として [基本機能 + ユーザー ID のアクセストークンのカスタマイズ] を選択します。V3_0 イベントを生成するには、[基本機能 + ユーザー ID とマシン ID のアクセストークンのカスタマイズ] を選択します。[UpdateUserPool](#) API リクエストまたは [CreateUserPool](#) API リクエストの [LambdaConfig](#) パラメータで `LambdaVersion` の値を設定することもできます。イベントバージョン 1、2、3 は、エッセンシャル機能プランとプラス機能プランで利用できます。バージョン 3 イベントの M2M オペレーションには、月間アクティブユーザー (MAU) の計算式とは別の料金体系があります。詳細については、「[Amazon Cognito の料金](#)」を参照してください。

Note

ユーザープールが 2024 年 11 月 22 日 18:00 GMT 以前に [高度なセキュリティ機能] オプションを使用して動作し、ライト機能階層に残っている場合は、トークン生成前トリガーのイベントバージョン 1 と 2 にアクセスできます。このレガシー階層のユーザープールで高度なセキュリティ機能を有効にしていない場合は、イベントバージョン 1 にアクセスできます。バージョン 3 は、エッセンシャルとプラスでのみ利用できます。

クレームとスコープのリファレンス

Amazon Cognito は、アクセストークンと ID トークンに対して追加、変更、または抑制できるクレームとスコープを制限します。次の表は、Lambda 関数で変更できるクレームと変更できないクレーム、およびクレームの存在や値に影響するトリガーイベントパラメータを示しています。

Claim	デフォルトのトークンタイプ	追加可能?	変更可能?	抑制可能?	イベントパラメータ - 追加または変更	イベントパラメータ - 抑制	ID のタイプ	イベントバージョン
ユーザープールのトークンスキーマにないクレーム	なし	はい	はい	該当なし	claimsToOverride	claimsToSuppress	ユーザー、マシン ¹	すべて ²
scope	アクセス	はい	はい	はい	scopesToOverride	scopesToSuppress	ユーザー、マシン ¹	v2_0, v3_0
cognito:groups	ID、アクセス	はい	はい	はい	groupsToOverride	claimsToSuppress	ユーザー	すべて ²
cognito:preferred_role	ID	はい	はい	はい	preferredRole	claimsToSuppress	ユーザー	すべて
cognito:roles	ID	はい	はい	はい	iamRolesToOverride	claimsToSuppress	ユーザー	すべて
cognito:username	ID	いいえ	なし	いいえ	該当なし	該当なし	ユーザー	該当なし
cognito:プレフィック	なし	いいえ	なし	いいえ	該当なし	該当なし	該当なし	該当なし

Claim	デフォルトのトークンタイプ	追加可能?	変更可能?	抑制可能?	イベントパラメータ - 追加または変更	イベントパラメータ - 抑制	ID のタイプ	イベントバージョン
スが付いたその他のクレーム								
username	アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー	v2_0, v3_0
sub	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー	該当なし
標準 OIDC 属性	ID	はい	はい	はい	claimsTo: dd0r0ver: ide	claimsTo: uppress	ユーザー	すべて
custom: 属性	ID	はい	はい	はい	claimsTo: dd0r0ver: ide	claimsTo: uppress	ユーザー	すべて
dev: 属性	ID	いいえ	なし	はい	該当なし	claimsTo: uppress	ユーザー	すべて
identities	ID	いいえ	なし	いいえ	該当なし	該当なし	ユーザー	該当なし
aud ⁴	ID	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし

Claim	デフォルトのトークンタイプ	追加可能?	変更可能?	抑制可能?	イベントパラメータ - 追加または変更	イベントパラメータ - 抑制	ID のタイプ	イベントバージョン
client_id	アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
event_id	アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
device_key	アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー	該当なし
version	アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
acr	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
amr	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
at_hash	ID	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし

Claim	デフォルトのトークンタイプ	追加可能?	変更可能?	抑制可能?	イベントパラメータ - 追加または変更	イベントパラメータ - 抑制	ID のタイプ	イベントバージョン
auth_time	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
azp	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
exp	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
iat	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
iss	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
jti	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
nbf	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし

Claim	デフォルトのトークンタイプ	追加可能?	変更可能?	抑制可能?	イベントパラメータ - 追加または変更	イベントパラメータ - 抑制	ID のタイプ	イベントバージョン
nonce	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
origin_jti	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし
token_use	ID、アクセス	いいえ	なし	いいえ	該当なし	該当なし	ユーザー、マシン	該当なし

¹ マシン ID のアクセストークンは、トリガー入力イベントの v3_0 でのみ使用できます。イベントバージョン 3 は、エッセンシャル機能階層とプラス機能階層でのみ使用できます。ライト階層のユーザープールは v1_0 イベントを受信できます。高度なセキュリティ機能を備えた、ライト階層のユーザープールは、v1_0 イベントと v2_0 イベントを受信できます。

² トークン生成前トリガーを、v1_0 イベントバージョン (ID トークンのみ)、v2_0 イベントバージョン (ID トークンとアクセストークン)、v3_0 イベントバージョン (ID トークンとアクセストークン、さらにマシン ID の機能) に設定します。

³ cognito:preferred_role クレームと cognito:roles クレームを抑制するには、cognito:groups を claimsToSuppress に追加します。

⁴ aud クレームをアクセストークンに追加することはできますが、その値は現在のセッションのアプリケーションクライアント ID と一致する必要があります。リクエストイベントのクライアント ID は、event.callerContext.clientId から取得できます。

ID トークンのカスタマイズ

トークン生成前 Lambda トリガーのすべてのイベントバージョンでは、ユーザープールのアイデンティティ (ID) トークンの内容をカスタマイズできます。ID トークンは、ウェブまたはモバイルアプリにサインインするためのユーザー属性を、信頼できる ID ソースから提供します。ID トークンの詳細については、「[ID トークンの理解](#)」を参照してください。

トークン生成前の Lambda トリガーでは、ID トークンを使用して以下のような操作を行います。

- ユーザーがアイデンティティプールにリクエストした IAM ロールをランタイムに変更します。
- 外部ソースからユーザー属性を追加します。
- 既存のユーザー属性値を追加または置換します。
- アプリに渡されるはずのユーザー属性が、ユーザーの許可されたスコープや、アプリクライアントに付与した属性の読み取りアクセス権により開示されることを抑制します。

アクセストークンのカスタマイズ

トークン生成前 Lambda トリガーのイベントバージョン 2 および 3 を使用すると、ユーザープールのアクセストークンの内容をカスタマイズできます。アクセストークンは、Amazon Cognito トークン認可 API オペレーションやサードパーティ API などのアクセス保護されたリソースから情報を取得することをユーザーに許可します。クライアント認証情報の付与による Machine to Machine (M2M) 認証の場合、Amazon Cognito は、ユーザープールがバージョン 3 (V3_0) イベント用に設定されている場合にのみ、トークン生成前トリガーを呼び出します。アクセストークンの詳細については、「[アクセストークンの理解](#)」を参照してください。

トークン生成前の Lambda トリガーでは、アクセストークンを使用して以下のような操作を行います。

- scope クレーム内のスコープを追加または抑制します。例えば、スコープ `aws.cognito.signin.user.admin` のみを割り当てる Amazon Cognito ユーザープール API 認証で生成されたアクセストークンにスコープを追加できます。
- ユーザープールグループのユーザーのメンバーシップを変更します。
- Amazon Cognito アクセストークンにまだ存在していないクレームを追加します。
- アプリに渡されるはずのクレームの開示を抑制します。

ユーザープールでのアクセスのカスタマイズをサポートするには、トリガーリクエストの更新バージョンを生成するようにユーザープールを設定する必要があります。ユーザープールを更新するには、次の手順に従います。

AWS マネジメントコンソール

トークン生成前の Lambda トリガーでアクセストークンのカスタマイズをサポートするには

1. [Amazon Cognito コンソール](#)に移動し、[ユーザープール] を選択します。
2. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [拡張機能] メニューを選択し、[Lambda トリガー] を見つけます。
4. トークン生成前トリガーを追加または編集します。
5. [Lambda 関数を割り当てる] で Lambda 関数を選択します。
6. [トリガーイベントバージョン] として、[基本機能 + ユーザー ID のアクセストークンのカスタマイズ] または [機能機能 + ユーザー ID とマシン ID のアクセストークンのカスタマイズ] を選択します。この設定により、Amazon Cognito が関数に送信するリクエストパラメータが更新され、アクセストークンをカスタマイズするためのフィールドが含まれるようになります。

User pools API

トークン生成前の Lambda トリガーでアクセストークンのカスタマイズをサポートするには

[CreateUserPool](#) または [UpdateUserPool](#) API リクエストを生成します。デフォルト値に設定しないすべてのパラメータには、値を指定する必要があります。詳細については、「[ユーザープールとアプリケーションクライアントの設定更新](#)」を参照してください。

リクエストの LambdaVersion パラメータに以下の内容を含めます。LambdaVersion の値が V2_0 である場合、ユーザープールはアクセストークンのパラメータを追加し、アクセストークンに変更を適用します。LambdaVersion の値が V3_0 である場合、V2_0 と同じイベントが生成されますが、ユーザープールは M2M アクセストークンに対しても変更を適用します。特定の関数バージョンを呼び出すには、関数バージョンを LambdaArn の値とする Lambda 関数 ARN を使用します。

```
"PreTokenGenerationConfig": {  
  "LambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:MyFunction",  
  "LambdaVersion": "V3_0"
```



```
},
```

Machine to Machine (M2M) クライアント認証情報のクライアントメタデータ

[クライアントメタデータ](#)は M2M リクエストで渡すことができます。クライアントメタデータは、[トークン生成前の Lambda トリガー](#)の結果に役立つユーザーまたはアプリケーション環境からの追加情報です。ユーザープリンシパルによる認証オペレーションでは、[AdminRespondToAuthChallenge](#) API リクエストおよび [RespondToAuthChallenge](#) API リクエストの本文で、クライアントメタデータをトークン生成前トリガーに渡すことができます。アプリケーションは、[トークンエンドポイント](#)への直接リクエストを使用して M2M のアクセストークン生成フローを実行するため、異なるモデルとなります。クライアント認証情報のトークンリクエストの POST 本文で、クライアントメタデータオブジェクトを URL エンコード (x-www-form-urlencoded) した `aws_client_metadata` パラメータを文字列に渡します。リクエスト例については、「[基本認可によるクライアント認証情報](#)」を参照してください。次に示すのは、キーと値のペア `{"environment": "dev", "language": "en-US"}` を渡すパラメータの例です。

```
aws_client_metadata=%7B%22environment%22%3A%20%22dev%22,%20%22language%22%3A%20%22en-US%22%7D
```

その他のリソース

- [Amazon Cognito ユーザープールでアクセストークンをカスタマイズする方法](#)

トピック

- [トークン生成前の Lambda トリガーのソース](#)
- [トークン生成前の Lambda トリガーのパラメータ](#)
- [トークン生成前トリガーイベントバージョン 2 の例:クレーム、スコープ、グループの追加と非表示](#)
- [トークン生成前イベントバージョン 2 の例: 複雑なオブジェクトでクレームを追加する](#)
- [トークン生成前イベントバージョン 1 の例: 新しいクレームの追加と既存のクレームの非表示](#)
- [トークン生成前イベントバージョン 1 の例: ユーザーグループのメンバーシップの変更](#)

トークン生成前の Lambda トリガーのソース

triggerSource 値	イベント
TokenGeneration_HostedAuth	Amazon Cognito マネージドログインのサインインページからの認証時に呼び出されます。
TokenGeneration_Authentication	ユーザー認証フローが完了した後に呼び出されます。
TokenGeneration_NewPasswordChallenge	管理者によってユーザーが作成された後に呼び出されます。このフローは、ユーザーが一時パスワードを変更する必要があるときに呼び出されます。
TokenGeneration_ClientCredentials	M2M クライアント認証情報の付与の後に呼び出されます。ユーザープールがこのイベントを送信するのは、イベントバージョンが V3_0 の場合のみです。
TokenGeneration_AuthenticateDevice	ユーザーデバイスの認証の終了時に呼び出されます。
TokenGeneration_RefreshTokens	ユーザーが ID およびアクセスのトークンを更新しようとしたときに呼び出されます。

トークン生成前の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。トークン生成前の Lambda トリガーをユーザープールに追加するときに、トリガーバージョンを選択できます。このバージョンにより、Amazon Cognito がアクセストークンをカスタマイズするための追加パラメータを含むリクエストを Lambda 関数に渡すかどうかが決まります。

Version one

バージョン 1 のトークンは、ID トークン内のグループメンバーシップ、IAM ロール、新しいクレームを設定できます。グループメンバーシップのオーバーライドは、アクセストークンの `cognito:groups` クレームにも適用されます。

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

Versions two and three

バージョン 2 および 3 のリクエストイベントは、アクセストークンをカスタマイズするフィールドを追加します。ユーザープールは、バージョン 3 のイベントの変更をマシン ID のアクセストークンに適用します。また、これらのバージョンは、レスポンスオブジェクトの複雑な `claimsToOverride` データ型のサポートも追加します。Lambda 関数は、次のタイプのデータを `claimsToOverride` の値で返すことができます。

- String
- Number
- ブール値
- 文字列、数値、ブール値、またはこれらの組み合わせの配列
- 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"
  }
}
}
```

トークン生成前のリクエストパラメータ

名前	説明	トリガーイベントの最小バージョン
userAttributes	ユーザープール内のユーザープロフィールの属性。	1
groupConfiguration	現在のグループ設定を含む入力オブジェクト。このオブジェクトには、groupsToOverride 、 iamRolesToOverride 、 および preferredRole が含まれています。	1
groupsToOverride	ユーザーがメンバーになっている ユーザープールグループ 。	1
iamRolesToOverride	ユーザープールグループを AWS Identity and Access Management (IAM) ロールに関連付けることができます。この要素は、ユーザーがメンバーになっているグループのすべての IAM ロールのリストです。	1
preferredRole	ユーザープールグループには、 優先順位 を設定できます。この要素には、groupsToOverride 要素内で優先順位が最も高いグループの IAM ロールの名前が含まれます。	1

名前	説明	トリガーイベントの最小バージョン
clientMetadata	トークン生成前のトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。 このデータを Lambda 関数に渡すには、AdminRespondToAuthChallenge および RespondToAuthChallenge API オペレーションで ClientMetadata パラメータを使用します。Amazon Cognito は、トークン生成前の関数に渡すリクエストの AdminInitiateAuth および InitiateAuth API オペレーションに ClientMetadata パラメータのデータを含めません。	1
scopes	アクセストークンのスコープ。アクセストークンに含まれるスコープは、ユーザーがリクエストし、アプリケーションに発行を許可したユーザープールの標準スコープとカスタムスコープです。	2

トークン生成前のレスポンスパラメータ

名前	説明	トリガーイベントの最小バージョン
claimsOverrideDetails	A container for all elements in a V1_0 trigger event.	1
claimsAndScopeOverrideDetails	V2_0 または V3_0 トリガーイベント内のすべての要素を格納するコンテナ。	2
idTokenGeneration	ユーザーの ID トークンで上書き、追加、または抑制するクレーム。この ID トークンのカスタマイズ値の親は、	2

名前	説明	トリガーイベントの最小バージョン
	バージョン 2 以降でのみ表示されますが、子要素はバージョン 1 のイベントでも表示されます。	
accessTokenGeneration	ユーザーのアクセストークンで上書き、追加、または抑制するクレームとスコープ。このアクセストークンのカスタマイズ値の親は、バージョン 2 以降のイベントでのみ表示されます。	2
claimsToAddOrOverride	追加または変更する 1 つ以上のクレームとその値のマップ。グループ関連のクレームには、代わりに groupOverrideDetails を使用します。 イベントバージョン 2 以降の場合、この要素は accessTokenGeneration と idTokenGeneration の両方の下に表示されます。	1*
claimsToSuppress	Amazon Cognito で抑制するクレームのリスト。関数がクレーム値の非表示と置換の両方を行う場合、Amazon Cognito はクレームを非表示にします。 イベントバージョン 2 以降の場合、この要素は accessTokenGeneration と idTokenGeneration の両方の下に表示されます。	1

名前	説明	トリガーイベントの最小バージョン
groupOverrideDetails	現在のグループ設定を含む出力オブジェクト。このオブジェクトには、groupsToOverride 、iamRolesToOverride 、および preferredRole が含まれています。 この関数は、groupOverrideDetails オブジェクトを、指定したオブジェクトに置き換えます。レスポンスで空または Null オブジェクトを返すと、Amazon Cognito はグループを非表示にします。既存のグループ設定を同じままにするには、リクエストの groupConfiguration オブジェクトの値をレスポンスの groupOverrideDetails オブジェクトにコピーします。その後、サービスに渡します。 Amazon Cognito ID とアクセストークンには、両方とも cognito:groups クレームが含まれています。groupOverrideDetails オブジェクトは、アクセストークンおよび ID トークンの cognito:groups クレームを置き換えます。バージョン 1 のイベントがアクセストークンを変更できるのは、グループのオーバーライドのみです。	1
scopesToAdd	ユーザーのアクセストークンの scope クレームに追加するスコープのリスト。1 つ以上の空白文字を含むスコープ値を追加することはできません。	2
scopesToSuppress	ユーザーのアクセストークンの scope クレームから削除するスコープのリスト。	2

* バージョン 1 のイベントへのレスポンスオブジェクトは、文字列を返すことができます。バージョン 2 および 3 のイベントへのレスポンスオブジェクトは、[複雑なオブジェクト](#)を返すことができます。

トークン生成前トリガーイベントバージョン 2 の例: クレーム、スコープ、グループの追加と非表示

この例では、ユーザーのトークンに以下の変更を行います。

1. ID トークンに `family_name` として `Doe` を設定します。
2. ID トークンに `email` クレームと `phone_number` クレームが表示されないようにします。
3. ID トークンの `cognito:roles` クレームを `"arn:aws:iam::123456789012:role\sns_callerA", "arn:aws:iam::123456789012:role\sns_callerC", "arn:aws:iam::123456789012:role\sns_callerB"` に設定します。
4. ID トークンの `cognito:preferred_role` クレームを `arn:aws:iam::123456789012:role/sns_caller` に設定します。
5. スコープとして `openid`、`email`、`solar-system-data/asteroids.add` をアクセストークンに追加します。
6. アクセストークンの `phone_number` スコープと `aws.cognito.signin.user.admin` スコープを非表示にします。`phone_number` を削除すると、`userInfo` からユーザーの電話番号が取得されなくなります。`aws.cognito.signin.user.admin` を削除すると、ユーザーが Amazon Cognito ユーザープール API を使用して自分のプロフィールを読み取ったり変更したりするための API リクエストが禁止されます。

Note

`phone_number` をスコープから削除した場合、アクセストークンの残りのスコープに `openid` と少なくとも 1 つの標準スコープが含まれていれば、ユーザーの電話番号のみが取得できなくなります。詳細については、「[スコープについて](#)」を参照してください。

7. ID トークンとアクセストークンの `cognito:groups` クレームを `"new-group-A", "new-group-B", "new-group-C"` に設定します。

JavaScript

```
export const handler = function(event, context) {
  event.response = {
    "claimsAndScopeOverrideDetails": {
      "idTokenGeneration": {
        "claimsToAddOrOverride": {
          "family_name": "Doe"
        }
      }
    }
  }
}
```

```
    },
    "claimsToSuppress": [
        "email",
        "phone_number"
    ]
},
"accessTokenGeneration": {
    "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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

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": []
  }
}
```

トークン生成前イベントバージョン 2 の例: 複雑なオブジェクトでクレームを追加する

この例では、ユーザーのトークンに以下の変更を行います。

1. ID トークンに数値、文字列、ブール値、JSON タイプのクレームを追加します。これは、ID トークンでバージョン 2 のトリガーイベントが利用できる唯一の変更です。
2. アクセストークンに数値、文字列、ブール値、JSON タイプのクレームを追加します。
3. アクセストークンに 3 つのスコープを追加します。
4. ID トークンとアクセストークンの email クレームを抑制します。
5. アクセストークンの `aws.cognito.signin.user.admin` スコープを抑制します。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

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": []
  }
}
```

トークン生成前イベントバージョン 1 の例: 新しいクレームの追加と既存のクレームの非表示

この例では、トークン生成前の Lambda 関数でバージョン 1 のトリガーイベントを使用して、新しいクレームを追加し、既存のクレームを抑制します。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。コード例 ではリクエストパラメータを処理しないため、空のリクエストでテストイベントを使用できます。共通のリクエストパラメータの詳細については、「[ユーザープールの Lambda トリガーイベント](#)」を参照してください。

JSON

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

トークン生成前イベントバージョン 1 の例: ユーザーグループのメンバーシップの変更

この例では、トークン生成前の Lambda 関数でバージョン 1 のトリガーイベントを使用し、ユーザーのグループメンバーシップを変更します。

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::XXXXXXXXXXXX:role/sns_callerC",
        ],
        preferredRole: "arn:aws:iam::XXXXXXXXXXXX:role/sns_caller",
      },
    },
  },
};

return event;
};

export { handler };
```

Amazon Cognito は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

JSON

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

ユーザー移行の Lambda トリガー

パスワードによるサインイン時にユーザーがユーザープールに存在しない場合、またはユーザーがパスワードを忘れた場合のフローにいる場合に、Amazon Cognito は、このトリガーを呼び出します。Amazon Cognito は、Lambda 関数が正常に値を返した後でユーザーをユーザープールに作成し

ます。ユーザー移行の Lambda トリガーを使用した認証フローの詳細については、「[ユーザー移行の Lambda トリガーを使用したユーザーのインポート](#)」を参照してください。

サインイン時、またはパスワードを忘れた場合のフロー中に、ユーザーを既存のユーザーディレクトリから Amazon Cognito ユーザープールに移行するには、この Lambda トリガーを使用します。

トピック

- [ユーザー移行の Lambda トリガーのソース](#)
- [ユーザー移行の Lambda トリガーのパラメータ](#)
- [例: 既存のパスワードを使用したユーザーの移行](#)

ユーザー移行の Lambda トリガーのソース

triggerSource 値	イベント
UserMigration_Authentication ¹	サインイン時のユーザーの移行。
UserMigration_ForgotPassword	パスワードを忘れた場合のフロー実行時のユーザー移行

¹ ユーザーが[パスワードなしのサインイン](#)で認証する場合、Amazon Cognito はこのトリガーを呼び出しません。

ユーザー移行の Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

```
{
  "userName": "string",
  "request": {
    "password": "string",
    "validationData": {
      "string": "string",
      . . .
    }
  }
}
```

```
    },
    "clientMetadata": {
      "string": "string",
      . . .
    }
  },
  "response": {
    "userAttributes": {
      "string": "string",
      . . .
    },
    "finalUserStatus": "string",
    "messageAction": "string",
    "desiredDeliveryMediums": [ "string", . . . ],
    "forceAliasCreation": boolean,
    "enableSMSMFA": boolean
  }
}
```

ユーザー移行リクエストパラメータ

userName

ユーザーがサインイン時に入力するユーザー名。

password

ユーザーがサインイン時に入力するパスワード。Amazon Cognito は、パスワードを忘れた場合のフローによって開始されたリクエストでこの値を送信しません。

validationData

ユーザーサインインリクエストに検証データを含む 1 つ以上のキーバリューペア。このデータを Lambda 関数に渡すには、[InitiateAuth](#) および [AdminInitiateAuth](#) API アクションで ClientMetadata パラメータを使用できます。

clientMetadata

ユーザー移行のトリガーの Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキーバリューペア。このデータを Lambda 関数に渡すには、[AdminRespondToAuthChallenge](#) および [ForgotPassword](#) API アクションで ClientMetadata パラメータを使用できます。

ユーザー移行レスポンスパラメータ

userAttributes

このフィールドは必須です。

このフィールドには、Amazon Cognito がユーザープール内のユーザープロフィールに保存され、ユーザー属性として使用される、名前と値のペアが 1 つ以上含まれている必要があります。標準およびカスタムの両方のユーザー属性を含めることができます。カスタム属性は、標準属性と区別するために、custom: プレフィックスを必要とします。詳細については、「[カスタム属性](#)」を参照してください。

Note

パスワードを忘れた場合のフローでパスワードをリセットするには、ユーザーに検証済みの E メールアドレスまたは電話番号のどちらかが必要です。Amazon Cognito は、ユーザー属性にある E メールアドレスまたは電話番号宛てに、パスワードのリセットコードが含まれるメッセージを送信します。

属性	要件
ユーザープールの作成時に必須とマークした属性	移行時に必須属性が見当たらない場合は、Amazon Cognito はデフォルト値を使用します。
username	サインイン用のユーザー名に加えてエイリアス属性を使用してユーザープールを設定しており、ユーザーがユーザー名として有効なエイリアスを入力している場合は、必須です。エイリアスの値は、E メールアドレス、優先ユーザー名、電話番号です。 リクエストとユーザープールがエイリアスの要件を満たしている場合、関数からのレスポンスは、受け取った username パラメータをエイリアス属性に割り当てる必要があります。また、レスポンスは username 属性に自身の値を割り当てる必要があります。ユーザープールが、受け取った username をエイリアスにマッピングするのに必要な条件を満たしていない場合、レスポンスの username パラメータはリクエストと完全に一致するか、省略する必要があります。

属性	要件
	 Note username はユーザープールで一意である必要があります。

finalUserStatus

このパラメータを CONFIRMED に設定すると、ユーザーが以前のパスワードでサインインできるように自動確認することができます。ユーザーを CONFIRMED に設定した場合は、サインインする前に追加のアクション操作を実行する必要はありません。この属性を CONFIRMED に設定しない場合は、RESET_REQUIRED に設定されます。

RESET_REQUIRED の finalUserStatus は、サインイン時の移行後、ユーザーはすぐにパスワードを変更する必要があり、クライアントアプリは認証フロー中に PasswordResetRequiredException を処理する必要があることを意味します。

Note

Amazon Cognito は、Lambda トリガーを使用して移行中にユーザープールに設定したパスワード強度ポリシーを強制しません。パスワードが設定したパスワードポリシーを満たさない場合でも、Amazon Cognito は引き続きユーザーを移行できるように、パスワードを受け入れます。パスワード強度ポリシーを施行してポリシーを満たさないパスワードを拒否するには、コード内のパスワード強度を検証して、パスワードがポリシーを満たさない場合は、finalUserStatus を RESET_REQUIRED に設定します。

messageAction

このパラメータを SUPPRESS に設定すると、通常、Amazon Cognito は新規ユーザーに送信するウェルカムメッセージの送信を拒否することができます。関数がこのパラメータを返さない場合、Amazon Cognito はウェルカムメッセージを送信します。

desiredDeliveryMediums

このパラメータを EMAIL に設定すると E メールで、SMS に設定すると SMS でウェルカムメッセージを送信できます。関数がこのパラメータを返さない場合、Amazon Cognito は SMS でウェルカムメッセージを送信します。

forceAliasCreation

このパラメータを TRUE に設定した場合、UserAttributes パラメータの E メールアドレスまたは電話番号がすでに別のユーザーのエイリアスとして存在していると、API の呼び出しで以前のユーザーのエイリアスが新しく作成されたユーザーに移行されます。以前のユーザーはそのエイリアスを使用してログインできなくなります。

このパラメータを、FALSE に設定し、エイリアスが存在する場合、Amazon Cognito はユーザーを移行せず、クライアントアプリにエラーを返します。

このパラメータを返さない場合、Amazon Cognito はその値が「false」であると見なします。

enableSMSMFA

このパラメータを true に設定すると、移行したユーザーはサインインするために SMS テキストメッセージの多要素認証 (MFA) を完了することが必要になります。ユーザープールで MFA が有効になっている必要があります。リクエストパラメータのユーザー属性には、電話番号を含める必要があります。電話番号がないと、そのユーザーの移行は失敗します。

例: 既存のパスワードを使用したユーザーの移行

この例の Lambda 関数は、既存のパスワードを使用してユーザーを移行し、Amazon Cognito からのウェルカムメッセージを抑制します。

Node.js

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

  if (event.triggerSource == "UserMigration_Authentication") {
    // authenticate the user with your existing user directory service
    user = authenticateUser(event.userName, event.request.password);
    if (user) {
      event.response.userAttributes = {
        email: user.emailAddress,
        email_verified: "true",
      };
      event.response.finalUserStatus = "CONFIRMED";
      event.response.messageAction = "SUPPRESS";
      context.succeed(event);
    } else {
```

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

カスタムメッセージの Lambda トリガー

ユーザーに送信する E メールメッセージと SMS メッセージの外部標準がある場合、または実行時に独自のロジックをユーザーメッセージのフォーマットに適用する場合は、カスタムメッセージトリガーをユーザープールに追加します。カスタムメッセージ Lambda は、ユーザープールが送信する前に、すべての E メールおよび SMS メッセージの内容を受信します。その後、Lambda 関数でメッセージの内容と件名を変更することができます。

Amazon Cognito は、E メールまたは電話による確認メッセージ、または多要素認証 (MFA) コードを送信する前にこのトリガーを呼び出します。カスタムメッセージトリガーを使用して、メッセージを動的にカスタマイズできます。

リクエストには `codeParameter` が含まれます。これは、Amazon Cognito がユーザーに配信するコードのプレースホルダーとなる文字列です。メッセージ本文で検証コードを表示する位置に `codeParameter` 文字列を挿入します。Amazon Cognito がこのレスポンスを受信すると、Amazon Cognito が `codeParameter` 文字列を実際の検証コードに置き換えます。

Note

CustomMessage_AdminCreateUser をトリガーソースとするカスタムメッセージ Lambda 関数の入力イベントには、ユーザー名と検証コードが含まれています。管理者が作成したユーザーはユーザー名とコードの両方を受け取る必要があるため、関数からのレスポンスにはユーザー名とコードのプレースホルダ変数を含める必要があります。メッセージのプレースホルダーは、request.usernameParameter および request.codeParameter の値です。これらの値は通常 {username} と {####} です。ベストプラクティスとして、変数名をハードコーディングするのではなく、入力値を参照します。

トピック

- [カスタムメッセージの Lambda トリガーのソース](#)
- [カスタムメッセージの Lambda トリガーのパラメータ](#)
- [サインアップのカスタムメッセージの例](#)
- [管理者作成ユーザーのカスタムメッセージの例](#)

カスタムメッセージの Lambda トリガーのソース

triggerSource 値	イベント
CustomMessage_SignUp	カスタムメッセージ - サインアップ後に確認コードを送信するため。
CustomMessage_AdminCreateUser	カスタムメッセージ - 新規ユーザーに一時パスワードを送信するため。
CustomMessage_ResendCode	カスタムメッセージ - 既存ユーザーに確認コードを再送するため。
CustomMessage_ForgotPassword	カスタムメッセージ - 忘れたパスワードのリクエスト用の確認コードを送信するため。
CustomMessage_UpdateUserAttribute	カスタムメッセージ - ユーザーの E メールまたは電話番号が変更されると、このトリガーは

triggerSource 値	イベント
	検証コードをそのユーザーに自動的に送信します。他の属性には使用できません。
CustomMessage_VerifyUserAttribute	カスタムメッセージ - ユーザーが手動で新しい E メールや電話番号の認証コードをリクエストすると、このトリガーからユーザーに認証コードが送信されます。
CustomMessage_Authentication	カスタムメッセージ - 認証時に MFA コードを送信するため。

カスタムメッセージの Lambda トリガーのパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する [共通パラメータ](#)を組み合わせたものです。

JSON

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


カスタムメッセージリクエストパラメータ

userAttributes

ユーザー属性を表す 1 つ以上の名前 - 値ペア。

codeParameter

カスタムメッセージで、検証コードのプレースホルダーとして使用する文字列。

usernameParameter

ユーザー名。Amazon Cognito は、管理者が作成したユーザーからのリクエストにこのパラメータを含めます。

clientMetadata

カスタムメッセージのトリガーに指定する Lambda 関数へのカスタム入力として提供できる 1 つ、または複数のキー/値ペア。カスタムメッセージ関数を呼び出すリクエストには、[AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションの ClientMetadata パラメータで渡されたデータは含まれません。このデータを Lambda 関数に渡すには、以下の API アクションで ClientMetadata パラメータを使用できます。

- [AdminResetUserPassword](#)
- [AdminRespondToAuthChallenge](#)
- [AdminUpdateUserAttributes](#)
- [ForgotPassword](#)
- [GetUserAttributeVerificationCode](#)
- [ResendConfirmationCode](#)
- [SignUp](#)
- [UpdateUserAttributes](#)

カスタムメッセージレスポンスパラメータ

レスポンスで、ユーザーへのメッセージに使用するカスタムテキストを指定します。これらのパラメータに対して Amazon Cognito が適用する文字列制約については、「[MessageTemplateType](#)」を参照してください。

smsMessage

ユーザーに送信されるカスタム SMS メッセージ。リクエストで受信する `codeParameter` 値を含める必要があります。

emailMessage

ユーザーに送信するカスタム E メールメッセージ。emailMessage パラメータで HTML 書式を使用できます。リクエストで受信される `codeParameter` 値を変数 {####} として含める必要があります。Amazon Cognito は、ユーザープールの EmailSendingAccount 属性が DEVELOPER の場合のみ、emailMessage パラメータを使用できます。ユーザープールの EmailSendingAccount 属性が、DEVELOPER ではなく、emailMessage パラメータが返されると、Amazon Cognito は 400 エラーコード `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException` を生成します。Amazon Simple Email Service (Amazon SES) を使用して E メールメッセージを送信する場合、ユーザープールの EmailSendingAccount 属性は DEVELOPER です。それ以外の場合、値は COGNITO_DEFAULT です。

emailSubject

カスタムメッセージの件名。emailSubject パラメータは、ユーザープールの EmailSendingAccount 属性が DEVELOPER の場合にのみ使用できます。ユーザープールの EmailSendingAccount 属性が DEVELOPER ではなく、Amazon Cognito が emailSubject パラメータを返した場合、Amazon Cognito は 400 エラーコード `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException` を生成します。Amazon Simple Email Service (Amazon SES) を使用して E メールメッセージを送信する場合、ユーザープールの EmailSendingAccount 属性は DEVELOPER です。それ以外の場合、値は COGNITO_DEFAULT です。

サインアップのカスタムメッセージの例

この Lambda 関数の例は、サービスでアプリがユーザーへの検証コードの送信が必要とされる場合に、E メールまたは SMS メッセージをカスタマイズします。

Amazon Cognito は、登録後、検証コードの再送信時、パスワードを忘れた場合、ユーザー属性の検証時といった複数のイベントで Lambda トリガーを呼び出すことができます。レスポンスには、SMS と Eメールの両方のメッセージが含まれます。メッセージにはコードのパラメータ "####" を含める必要があります。このパラメータは、ユーザーが受け取る検証コードのプレースホルダーです。

E メールメッセージの最大長は 20,000 UTF-8 文字です。この長さには検証コードが含まれます。これらの E メールメッセージでは、HTML タグを使用できます。

SMS メッセージの最大長は 140 UTF-8 文字です。この長さには検証コードが含まれます。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

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"
}
}
```

管理者作成ユーザーのカスタムメッセージの例

Amazon Cognito がこのカスタムメッセージ Lambda 関数のサンプルに送信したリクエストは、triggerSource 値が CustomMessage_AdminCreateUser で、ユーザー名と仮パスワードが含まれます。関数は、リクエストの仮パスワードから `${event.request.codeParameter}` を入力し、リクエストのユーザー名から `${event.request.usernameParameter}` を入力します。

カスタムメッセージは、レスポンスオブジェクトの smsMessage と emailMessage に codeParameter と usernameParameter の値を挿入する必要があります。この例では、関数は、レスポンスフィールド `event.response.smsMessage` と `event.response.emailMessage` に同じメッセージを書き込みます。

E メールメッセージの最大長は 20,000 UTF-8 文字です。この長さには検証コードが含まれます。これらの E メールでは HTML タグを使用できます。SMS メッセージの最大長は 140 UTF-8 文字です。この長さには検証コードが含まれます。

レスポンスには、SMS と Eメールの両方のメッセージが含まれます。

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 は Lambda 関数にイベント情報を渡します。関数はレスポンスで、同じイベントオブジェクトを変更と共に Amazon Cognito に返します。Lambda コンソールで、Lambda トリガーに関連するデータを使用したテストイベントをセットアップできます。以下は、このコードサンプルのテストイベントです。

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>"
  }
}
```

```
}
```

カスタム送信者の Lambda トリガー

Lambda トリガーである CustomEmailSender と CustomSMSSender は、ユーザープールでサードパーティーの E メール通知および SMS 通知をサポートします。Lambda 関数コード内からユーザーに通知を送信するには、希望の SMS プロバイダーと E メールプロバイダーを選択して使用することができます。Amazon Cognito が、招待、確認コード、検証コード、および仮パスワードをユーザーに送信する必要がある場合、イベントは、設定された Lambda 関数を有効化します。Amazon Cognito は、有効化された Lambda 関数にコードと一時パスワード (シークレット) を送信します。Amazon Cognito は、AWS KMS カスタマーマネージドキーとを使用してこれらのシークレットを暗号化します AWS Encryption SDK。AWS Encryption SDK は、汎用データの暗号化と復号に役立つクライアント側の暗号化ライブラリです。

[CustomEmailSender](#)

Amazon Cognito は、E メール通知をユーザーに送信するためにこのトリガーを呼び出します。

[CustomSMSSender](#)

Amazon Cognito は、SMS 通知をユーザーに送信するためにこのトリガーを呼び出します。

暗号化の概念

Amazon Cognito は、カスタム送信者トリガーに送信するイベントでユーザーのコードをプレーンテキストで送信しません。Lambda 関数は、イベント内のコードを復号する必要があります。次の概念は、ユーザーに配信できるコードを取得するために関数が必要である暗号化アーキテクチャです。

AWS KMS

AWS KMS は、AWS KMS キーを作成および制御するためのマネージドサービスです。これらのキーは、データを暗号化します。詳細については、「[AWS Key Management Serviceとは何ですか?](#)」を参照してください。

KMS キー

KMS キーは、暗号化キーの論理表現です。KMS キーには、キー ID、作成日、説明、キーステータスなどのメタデータが含まれます。KMS キーには、データの暗号化と復号に使用されるキーマテリアルも含まれています。詳細については、「[AWS KMS キーの削除](#)」を参照してください。

対称 KMS キー

対称 KMS キーは、暗号化されていない AWS KMS を終了しない 256 ビットの暗号化キーです。対称 KMS キーを使用するには、[呼び出す必要があります AWS KMS](#)。Amazon Cognito は対称キーを使用します。同じキーで暗号化と復号化が行われます。詳細については、「[KMS キーの削除](#)」を参照してください。

カスタム送信者 Lambda トリガーについて知っておくべきこと

- これらの Lambda トリガーを使用するようにユーザープールを設定するには、AWS CLI または SDK を使用します。これらの設定は、Amazon Cognito コンソールからは利用できません。

UpdateUserPool オペレーションは Lambda 設定を指定します。このオペレーションをリクエストする場合は、ユーザープールのすべてのパラメータおよび変更対象のパラメータが必要です。関連するすべてのパラメータを指定しない場合、Amazon Cognito は不足しているパラメータの値をデフォルトに設定します。次の CLI AWS の例に示すように、ユーザープールに追加または保持するすべての Lambda 関数のエントリを含めます。詳細については、「[ユーザープールとアプリケーションクライアントの設定更新](#)」を参照してください。

```
#Send this parameter in an 'aws cognito-idp update-user-pool' CLI command,
including any existing
#user pool configurations. This snippet also includes a pre sign-up trigger for
syntax reference. The pre sign-up trigger
#doesn't have a role in custom sender triggers.

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

リクエストで UpdateUserPool の JSON 本文を使用する場合、次の LambdaConfig スニペットはカスタム SMS および E メール送信者関数をリクエストに割り当てます。

```
"LambdaConfig": {
  "KMSKeyID": "arn:aws:kms:us-east-1:111122223333:key/a6c4f8e2-0c45-47db-925f-87854bc9e357",
  "CustomEmailSender": {
    "LambdaArn": "arn:aws:lambda:us-east-1:111122223333:function:MyFunction",
```

```
    "LambdaVersion": "V1_0"
  },
  "CustomSMSSender": {
    "LambdaArn": "arn:aws:lambda:us-east-1:111122223333:function:MyFunction",
    "LambdaVersion": "V1_0"
  }
}
```

- `update-user-pool` AWS CLI コマンドを使用してカスタム送信者 Lambda トリガーを削除するには、`CustomSMSSender` または `CustomEmailSender` パラメータを省略し `--lambda-config`、ユーザープールで使用する他のすべてのトリガーを含めます。

`UpdateUserPool` API リクエストを使用してカスタム送信者の Lambda トリガーを削除するには、ユーザープール設定の残りを含むリクエスト本文から `CustomSMSSender` または `CustomEmailSender` パラメータを省略します。

- Amazon Cognito は、ユーザーの一時パスワードで `<` (`<`) および `>` (`>`) などの予約文字を HTML エスケープします。これらの文字は、Amazon Cognito がカスタム E メール送信者機能に送信する一時パスワードには表示される場合がありますが、一時的な確認コードには表示されません。一時パスワードを送信するには、Lambda 関数がパスワードを復号した後、ユーザーにメッセージを送信する前に、これらの文字をアンエスケープする必要があります。

カスタム送信者 Lambda トリガーのアクティブ化

カスタムロジックを使用して SMS または E メールメッセージをユーザープールに送信するには、カスタム送信者トリガーを設定します。以下の手順では、カスタム SMS トリガー、カスタム E メールトリガー、またはその両方をユーザープールに割り当てます。カスタム送信者トリガーを追加すると、Amazon Cognito は、SMS または E メールメッセージを送信するデフォルトの動作ではなく、電話番号やワンタイムコードなどのユーザー属性を常に Lambda 関数に送信します。

1. AWS Key Management Service () で [対称暗号化キー](#) を作成します AWS KMS。Amazon Cognito は、シークレット (仮パスワード、検証コード、認証用のワンタイムパスワード、確認コード) を生成し、この KMS キーを使用してシークレットを [AWS Encryption SDK](#) で暗号化します。その後、Lambda 関数 AWS Encryption SDK で を使用してシークレットを復号し、プレーンテキストでユーザーに送信できます。
2. ユーザープールを作成または更新する IAM プリンシパルは、Amazon Cognito がコードを暗号化するために使用する KMS キーに対して 1 回限りの付与を作成します。このプリンシパルに KMS キーへの `CreateGrant` アクセス許可を付与します。この例の KMS キーポリシーを有効にするには、ユーザープールを更新する管理者が、IAM ロール

arn:aws:iam::111222333444:role/my-example-administrator-role で引き受けたロールセッションにサインインしている必要があります。

次のリソースベースのポリシーを、環境に合わせて変更し、KMS キーに適用します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::11122223333:role/my-example-administrator-  
role"
      },
      "Action": "kms:CreateGrant",
      "Resource": "arn:aws:kms:us-  
west-2:11122223333:key/1example-2222-3333-4444-999example",
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:userpool-id": "us-west-2_EXAMPLE"
        }
      }
    },
    {
      "Sid": "Allow Lambda to decrypt",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::11122223333:role/my-lambda-function-role"
      },
      "Action": "kms:Decrypt",
      "Resource": "*"
    }
  ]
}
```

3. カスタム送信者トリガーのための Lambda 関数を作成します。Amazon Cognito は [AWS 暗号化 SDK](#) を使用してシークレット (一時パスワードまたはユーザーの API リクエストを承認するコード) を暗号化します。
 - a. KMS キーに対して少なくとも kms:Decrypt アクセス許可を持つ [Lambda 実行ロール](#) を割り当てます。

- b. メッセージを送信するための Lambda 関数コードを作成します。関数への入カイベントにはシークレットが含まれています。関数で、を使用してシークレットを復号 AWS Encryption SDK し、関連するメタデータを処理します。次に、メッセージを配信するカスタム API に、コード、独自のカスタムメッセージ、宛先の電話番号を送信します。
- c. Lambda 関数 AWS Encryption SDK に を追加します。詳細については、「[AWS Encryption SDK プログラミング言語](#)」を参照してください。Lambda パッケージを更新するには、次のステップを完了します。
 - i. Lambda 関数を .zip ファイルとして AWS マネジメントコンソールにエクスポートします。
 - ii. 関数を開き、 を追加します AWS Encryption SDK。詳細とダウンロードリンクについては、AWS Encryption SDK デベロッパーガイドの「[AWS Encryption SDK プログラミング言語](#)」を参照してください。
 - iii. SDK の依存関係を使用して関数を ZIP 圧縮し、その関数を Lambda にアップロードします。詳細については、「AWS Lambda デベロッパーガイド」の「[Lambda 関数の .zip ファイルアーカイブとしてのデプロイ](#)」を参照してください。
4. Amazon Cognito サービスプリンシパルに、Lambda 関数を呼び出すための cognito-idp.amazonaws.com へのアクセス権を付与します。

次の AWS CLI コマンドは、Lambda 関数を呼び出すためのアクセス許可を Amazon Cognito に付与します。

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

5. カスタム送信者 Lambda トリガーを追加する LambdaConfig パラメータを使用して [UpdateUserPool](#) API リクエストを生成します。Amazon Cognito コンソールでは、このタイプのトリガーを追加することはできません。カスタム送信者トリガーには、KMSKeyID に加えて、CustomSMSSender または CustomEmailSender (あるいは両方) の LambdaConfig パラメータが必要です。

カスタム E メール送信者の Lambda トリガー

ユーザープールにカスタム E メール送信者トリガーを割り当てると、ユーザーイベントで E メールメッセージの送信が必要になった時点で、Amazon Cognito がデフォルトの動作ではなく Lambda 関数を呼び出します。カスタム送信者トリガーを使用すると、AWS Lambda 関数は選択した方法とプ

ロバイダーを通じてユーザーに E メール通知を送信できます。関数のカスタムコードは、すべての e メールメッセージを、ユーザープールから処理して配信する必要があります。

このトリガーは、ユーザープールが E メールメッセージを送信する方法をより細かく制御するシナリオに役立ちます。Lambda 関数は、例えば、複数の検証済み ID またはクロス AWS リージョンを管理する場合など、Amazon SES API オペレーションへの呼び出しをカスタマイズできます。また、関数は、メッセージを別の配信メディアまたはサードパーティーサービスにリダイレクトすることもできます。

カスタム E メール送信者トリガーを設定する方法については、「[カスタム送信者 Lambda トリガーのアクティブ化](#)」を参照してください。

カスタム E メール送信者の Lambda トリガーのソース

以下の表には、Lambda コード内のカスタム E メールトリガーのソースに対するトリガーイベントが記載されています。

TriggerSource value	イベント
CustomEmailSender_SignUp	ユーザーがサインアップすると、Amazon Cognito がウェルカムメッセージを送信します。
CustomEmailSender_Authentication	ユーザーがサインインすると、Amazon Cognito が E メール OTP または MFA コードを送信します。
CustomEmailSender_ForgotPassword	ユーザーがパスワードをリセットするコードを要求します。
CustomEmailSender_ResendCode	ユーザーが代替アカウント確認コードをリクエストします。
CustomEmailSender_UpdateUserAttribute	ユーザーが E メールアドレスまたは電話番号の属性を更新すると、Amazon Cognito は属性を検証するためのコードを送信します。
CustomEmailSender_VerifyUserAttribute	ユーザーが新しい E メールアドレスまたは電話番号属性を作成し、Amazon Cognito は属性を検証するコードを送信します。

TriggerSource value	イベント
CustomEmailSender_AdminCreateUser	ユーザープールに新しいユーザーを作成すると、Amazon Cognito から一時パスワードが送信されます。
CustomEmailSender_AccountTakeOverNotification	Amazon Cognito は、ユーザーアカウントを引き継ぐ試みを検出し、ユーザーに通知を送信します。

カスタム E メール送信者の Lambda トリガーパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

```
{
  "request": {
    "type": "customEmailSenderRequestV1",
    "code": "string",
    "clientMetadata": {
      "string": "string",
      . . .
    },
    "userAttributes": {
      "string": "string",
      . . .
    }
  }
}
```

カスタム E メール送信者のリクエストパラメータ

型

リクエストバージョン。カスタム E メール送信者イベントの場合、この文字列の値は常に customEmailSenderRequestV1 です。

コード

関数が復号してユーザーに送信できる暗号化されたコード。

clientMetadata

カスタム E メール送信者 Lambda 関数トリガーへのカスタム入力として提供できる 1 つ以上のキーバリューペア このデータを Lambda 関数に渡すには、[AdminRespondToAuthChallenge](#) および [RespondToAuthChallenge](#) API アクションで ClientMetadata パラメータを使用します。Amazon Cognito は、認証後関数に渡すリクエストの [AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションの ClientMetadata パラメータからのデータを含めません。

Note

Amazon Cognito は、次のトリガーソースを持つイベントでカスタム E メールトリガー関数に ClientMetadata を送信します。

- CustomEmailSender_ForgotPassword
- CustomEmailSender_SignUp
- CustomEmailSender_Authentication

Amazon Cognito は、ソース CustomEmailSender_AccountTakeOverNotification を使用してトリガーイベントで ClientMetadata を送信しません。

userAttributes

ユーザー属性を表す 1 つ以上のキーバリューペア。

カスタム E メール送信者の応答パラメータ

Amazon Cognito は、カスタム E メール送信者のレスポンスに追加の情報が返されることを期待しません。Lambda 関数は、イベントを解釈し、コードを復号してから、メッセージコンテンツを配信する必要があります。一般的な関数は E メールメッセージをアSEMBルし、サードパーティーの SMTP リレーに誘導します。

コード例

次の Node.js 例は、カスタム E メール送信者の Lambda 関数で E メールメッセージイベントを処理します。この例では、関数に 2 つの環境変数が定義されていることを前提としています。

KEY_ID

ユーザーのコードの暗号化と復号に使用する KMS キーの ID。

KEY_ARN

ユーザーのコードを暗号化および復号化するために使用する KMS キーの Amazon リソースネーム (ARN)。

この関数をデプロイするには

1. デベロッパーワークスペースに最新バージョンの NodeJS をインストールします。
2. ワークスペースに新しい NodeJS プロジェクトを作成します。
3. `npm init -y` でプロジェクトを初期化します。
4. Lambda 関数のスクリプトを作成します (`touch index.mjs`)。
5. 以下の例の内容を `index.mjs` に貼り付けます。
6. プロジェクトの依存関係をダウンロードします AWS Encryption SDK。 `npm install @aws-crypto/client-node`
7. プロジェクトディレクトリをファイルに圧縮します (`zip -r my_deployment_package.zip .`)。
8. [ZIP ファイルを関数にデプロイします](#)。

このサンプル関数はコードを復号し、サインアップイベントの場合は、ユーザーの E メールアドレスへの E メールメッセージの送信をシミュレートします。

```
import { KmsKeyringNode, buildClient, CommitmentPolicy } from '@aws-crypto/client-node';

// Configure the encryption SDK client with the KMS key from the environment variables
const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT
);

const generatorKeyId = process.env.KEY_ID;
const keyIds = [process.env.KEY_ARN];
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds });

// Example function to simulate sending email.
// This example logs message details to CloudWatch Logs from your Lambda function.
```

```
// Update this function with custom logic that sends an email message to 'emailaddress'
// with body 'message'.
const sendEmail = async (emailAddress, message) => {
  // Log the destination with the email address masked.
  console.log(`Simulating email send to ${emailAddress.replace(/^[^@.]/g, '*')}`);
  // Log the message with the code masked.
  console.log(`Message content: ${message.replace(/\b\d{6,8}\b/g, '*****')}`);
  // Simulate API delay
  await new Promise(resolve => setTimeout(resolve, 100));
  console.log('Email sent successfully');
  return true;
};

export const handler = async (event) => {
  try {
    // Decrypt the secret code using encryption SDK
    let plainTextCode;
    if (event.request.code) {
      const { plaintext, messageHeader } = await decrypt(keyring,
Buffer.from(event.request.code, 'base64'));
      plainTextCode = Buffer.from(plaintext).toString('utf-8');
    }

    // Handle different trigger sources
    if (event.triggerSource == 'CustomEmailSender_SignUp') {
      const emailAddress = event.request.userAttributes.email;
      const message = `Welcome! Your verification code is: ${plainTextCode}`;
      await sendEmail(emailAddress, message);
    }
    else if (event.triggerSource == 'CustomEmailSender_ResendCode') {
      // Handle resend code
    }
    else if (event.triggerSource == 'CustomEmailSender_ForgotPassword') {
      // Handle forgot password
    }
    else if (event.triggerSource == 'CustomEmailSender_UpdateUserAttribute') {
      // Handle update attribute
    }
    else if (event.triggerSource == 'CustomEmailSender_VerifyUserAttribute') {
      // Handle verify attribute
    }
    else if (event.triggerSource == 'CustomEmailSender_AdminCreateUser') {
      // Handle admin create user
    }
  }
}
```

```
        else if (event.triggerSource == 'CustomEmailSender_Authentication') {
            // Handle authentication
        }
        else if (event.triggerSource ==
'CustomEmailSender_AccountTakeOverNotification') {
            // Handle account takeover notification
        }

        return;
    } catch (error) {
        console.error('Error in custom email sender:', error);
        throw error;
    }
};
```

カスタム SMS 送信者の Lambda トリガー

ユーザープールにカスタム SMS 送信者トリガーを割り当てると、ユーザーイベントで SMS メッセージの送信が必要になった時点で、Amazon Cognito がデフォルトの動作ではなく Lambda 関数を呼び出します。カスタム送信者トリガーを使用すると、AWS Lambda 関数は選択したメソッドとプロバイダーを通じて SMS 通知をユーザーに送信できます。関数のカスタムコードは、すべての SMS メッセージを、ユーザープールから処理して配信する必要があります。

このトリガーは、ユーザープールが SMS メッセージを送信する方法をより細かく制御するシナリオを提供します。Lambda 関数は、複数の発信元 ID またはクロス AWS リージョンを管理する場合など、Amazon SNS API オペレーションへの呼び出しをカスタマイズできます。また、関数は、メッセージを別の配信メディアまたはサードパーティーサービスにリダイレクトすることもできます。

カスタム E メール送信者トリガーを設定する方法については、「[カスタム送信者 Lambda トリガーのアクティブ化](#)」を参照してください。

カスタム SMS 送信者の Lambda トリガーのソース

以下の表には、Lambda コード内のカスタム SMS トリガーのソースに対するトリガーイベントが記載されています。

TriggerSource value	イベント
CustomSMSSender_SignUp	ユーザーがサインアップすると、Amazon Cognito がウェルカムメッセージを送信します。

TriggerSource value	イベント
CustomSMSSender_ForgotPassword	ユーザーがパスワードをリセットするコードを要求します。
CustomSMSSender_ResendCode	ユーザーは、登録を確認するために新しいコードをリクエストします。
CustomSMSSender_VerifyUserAttribute	ユーザーが新しい E メールアドレスまたは電話番号の属性を作成すると、Amazon Cognito は属性を検証するコードを送信します。
CustomSMSSender_UpdateUserAttribute	ユーザーが E メールアドレスまたは電話番号の属性を更新すると、Amazon Cognito は属性を検証するためのコードを送信します。
CustomSMSSender_Authentication	ユーザーがサインインすると、Amazon Cognito が SMS OTP または MFA コードを送信します。
CustomSMSSender_AdminCreateUser	ユーザープールに新しいユーザーを作成すると、Amazon Cognito から一時パスワードが送信されます。

カスタム SMS 送信者の Lambda トリガーパラメータ

Amazon Cognito がこの Lambda 関数に渡すリクエストは、以下のパラメータと Amazon Cognito がすべてのリクエストに追加する[共通パラメータ](#)を組み合わせたものです。

JSON

```
{
  "request": {
    "type": "customSMSSenderRequestV1",
    "code": "string",
    "clientMetadata": {
      "string": "string",
      . . .
    },
    "userAttributes": {
```

```
        "string": "string",  
        . . .  
    }  
}
```

カスタム SMS 送信者のリクエストパラメータ

型

リクエストバージョン。カスタム SMS 送信者イベントの場合、この文字列の値は常に `customSMSSenderRequestV1` です。

コード

関数が復号してユーザーに送信できる暗号化されたコード。

clientMetadata

カスタム SMS 送信者 Lambda 関数トリガーへのカスタム入力として提供できる 1 つ以上のキーバリュースタンプ。このデータを Lambda 関数に渡すには、[AdminRespondToAuthChallenge](#) および [RespondToAuthChallenge](#) API アクションで ClientMetadata パラメータを使用します。Amazon Cognito は、認証後関数に渡すリクエストの [AdminInitiateAuth](#) および [InitiateAuth](#) API オペレーションの ClientMetadata パラメータからのデータを含めません。

userAttributes

ユーザー属性を表す 1 つ以上のキーバリュースタンプ。

カスタム SMS 送信者の応答パラメータ

Amazon Cognito は、レスポンスに追加の返品情報を期待しません。関数では、API オペレーションを使用してリソースをクエリして変更したり、イベントメタデータを外部システムに記録することができます。

コード例

次の Node.js 例は、カスタム SMS 送信者 Lambda 関数で SMS メッセージイベントを処理します。この例では、関数に 2 つの環境変数が定義されていることを前提としています。

KEY_ID

ユーザーのコードの暗号化と復号に使用する KMS キーの ID。

KEY_ARN

ユーザーのコードを暗号化および復号化するために使用する KMS キーの Amazon リソースネーム (ARN)。

この関数をデプロイするには

1. デベロッパーワークスペースに最新バージョンの NodeJS をインストールします。
2. ワークスペースに新しい NodeJS プロジェクトを作成します。
3. `npm init -y` でプロジェクトを初期化します。
4. Lambda 関数のスクリプトを作成します (`touch index.mjs`)。
5. 以下の例の内容を `index.mjs` に貼り付けます。
6. プロジェクトの依存関係をダウンロードします AWS Encryption SDK。 `npm install @aws-crypto/client-node`
7. プロジェクトディレクトリをファイルに圧縮します (`zip -r my_deployment_package.zip .`)。
8. [ZIP ファイルを関数にデプロイします](#)。

```
import { KmsKeyringNode, buildClient, CommitmentPolicy } from '@aws-crypto/client-node';

// Configure the encryption SDK client with the KMS key from the environment variables
const { encrypt, decrypt } = buildClient(
  CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT
);

const generatorKeyId = process.env.KEY_ID;
const keyIds = [process.env.KEY_ARN];
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds });

// Example function to simulate sending SMS.
// This example logs message details to CloudWatch Logs from your Lambda function.
// Update this function with custom logic that sends an SMS message to 'phoneNumber'
// with body 'message'.
const sendSMS = async (phoneNumber, message) => {
  // Log the destination with the phone number masked.
  console.log(`Simulating SMS send to ${phoneNumber.replace(/[^\+]/g, '*')}`);
  // Log the message with the code masked.
```

```
    console.log(`Message content: ${message.replace(/\b\d{6,8}\b/g, '*****')}`);
    // Simulate API delay
    await new Promise(resolve => setTimeout(resolve, 100));
    console.log('SMS sent successfully');
    return true;
};

export const handler = async (event) => {
  try {
    // Decrypt the secret code using encryption SDK
    let plainTextCode;
    if (event.request.code) {
      const { plaintext, messageHeader } = await decrypt(keyring,
Buffer.from(event.request.code, 'base64'));
      plainTextCode = Buffer.from(plaintext).toString('utf-8');
    }

    // Handle different trigger sources
    if (event.triggerSource == 'CustomSMSSender_SignUp') {
      const phoneNumber = event.request.userAttributes.phone_number;
      const message = `Welcome! Your verification code is: ${plainTextCode}`;
      await sendSMS(phoneNumber, message);
    }
    else if (event.triggerSource == 'CustomSMSSender_ResendCode') {
      // Handle resend code
    }
    else if (event.triggerSource == 'CustomSMSSender_ForgotPassword') {
      // Handle forgot password
    }
    else if (event.triggerSource == 'CustomSMSSender_UpdateUserAttribute') {
      // Handle update attribute
    }
    else if (event.triggerSource == 'CustomSMSSender_VerifyUserAttribute') {
      // Handle verify attribute
    }
    else if (event.triggerSource == 'CustomSMSSender_AdminCreateUser') {
      // Handle admin create user
    }
    return;
  } catch (error) {
    console.error('Error in custom SMS sender:', error);
    throw error;
  }
}
```

```
};
```

トピック

- [カスタム SMS 送信者関数で SMS メッセージ機能を評価する](#)

カスタム SMS 送信者関数で SMS メッセージ機能を評価する

カスタム SMS 送信者 Lambda 関数は、ユーザープールが送信する SMS メッセージを受け入れ、関数はカスタムロジックに基づいてコンテンツを配信します。Amazon Cognito は関数に [カスタム SMS 送信者の Lambda トリガーパラメータ](#) を送信します。この情報を使用して、関数はユーザーが望むことを実行できます。例えば、Amazon Simple Notification Service (Amazon SNS) トピックにコードを送信できます。Amazon SNS トピックのサブスクライバーは、SMS メッセージ、HTTPS エンドポイント、または E メールアドレスです。

カスタム SMS 送信者 Lambda 関数を使用して Amazon Cognito SMS メッセージングのテスト環境を作成するには、[GitHub 上の aws-samples ライブラリ](#)の「[amazon-cognito-user-pool-development-and-testing-with-sms_redirected-to-email](#)」を参照してください。リポジトリには、新しいユーザープールを作成したり、既に持っているユーザープールを操作したりできる AWS CloudFormation テンプレートが含まれています。これらのテンプレートでは、Lambda 関数と Amazon SNS トピックが作成されます。テンプレートがカスタム SMS 送信者トリガーとして割り当てる Lambda 関数は、SMS メッセージを Amazon SNS トピックのサブスクライバーにリダイレクトします。

このソリューションをユーザープールにデプロイすると、Amazon Cognito が通常送信するすべてのメッセージが SMS メッセージングを介して送信され、代わりに Lambda 関数が中央の E メールアドレスに送信します。このソリューションを使用して、SMS メッセージをカスタマイズおよびプレビューし、Amazon Cognito が SMS メッセージを送信する原因となるユーザープールイベントをテストします。テストが完了したら、CloudFormation スタックをロールバックするか、ユーザープールからカスタム SMS 送信者関数の割り当てを削除します。

Important

[amazon-cognito-user-pool-development-and-testing-with-sms_redirected-to-email](#) のテンプレートは使用して、本番環境を構築しないでください。ソリューション内のカスタム SMS 送信者 Lambda 関数は SMS メッセージをシミュレートしますが、Lambda 関数はそれらすべてを単一の中央の E メールアドレスに送信します。本番環境の Amazon Cognito ユーザープールで SMS メッセージを送信する前に、[Amazon Cognito ユーザープール用の SMS メッセージ設定](#) に示す要件を満たす必要があります。

ユーザープール内のユーザーを管理する

ユーザープールを作成した後で、ユーザーアカウントの作成、確認、および管理を行うことができます。Amazon Cognito グループでは、IAM ロールをグループにマップすることによって、ユーザーと、ユーザーのリソースへのアクセスを管理できます。

Amazon Cognito ユーザープール内のユーザーを管理するには、さまざまな設定オプションと管理タスクが必要です。ユーザープールは数百万のユーザーにスケールできます。このスケールのユーザーディレクトリには、それと同様にスケーラブルで反復可能な管理ツールが必要です。多数のユーザープロフィールの作成、非アクティブなユーザーの管理、ガバナンスやコンプライアンスのレポート作成、ほとんどの作業をユーザーが行うセルフサービスツールの設定が必要になります。ユーザープールを作成したら、E メールや電話番号の検証を要求するなど、ユーザーのサインアップ方法とアカウントの確認方法を制御できます。また、管理者は、ユーザーアカウントの直接作成や、ウェルカムメッセージとパスワードの要件のカスタマイズもできます。

ユーザープールには、ユーザーのグループメンバーシップに基づいてリソースへのアクセスを管理できるユーザーグループが含まれます。これらのグループに IAM ロールを割り当てて、アイデンティティプールを使用して AWS のサービス へのアクセスを管理できます。ユーザーのグループメンバーシップは、ID トークンとアクセストークンの両方に存在します。この情報を使用して、アプリケーション内で、または Amazon Verified Permissions などのポリシーエンジンを用いて、実行時にアクセスコントロールを決定できます。

ユーザープールに多数のユーザーが含まれていることがよくあります。ユーザーアカウントを検索して更新する作業が多くなります。Amazon Cognito コンソールと API は、ユーザー名、E メール、電話番号などの標準属性に基づくユーザーのクエリをサポートします。管理者は、パスワードのリセット、アカウントの無効化、ユーザーイベント履歴の表示を行うこともできます。

既存のユーザーデータの移行については、Amazon Cognito には、CSV ファイルからユーザーをインポートするオプションや、ユーザーが初めてサインインしたときに [Lambda トリガー](#) を使用してユーザーを自動的に移行させるオプションがあります。これらのオプションは、他のユーザーディレクトリからユーザープールへのユーザー移行をサポートします。

ユーザープールのユーザー管理機能を使用して、ユーザーライフサイクルと認証エクスペリエンスをきめ細かく制御できます。セルフサービスサインアップ、管理者が作成したアカウント、グループ、移行ツールを組み合わせることで、Amazon Cognito ユーザープールは柔軟なユーザーディレクトリになります。

トピック

- [ユーザー作成ポリシーの設定](#)

- [ユーザーアカウントのサインアップと確認](#)
- [管理者としてのユーザーアカウントの作成](#)
- [ユーザープールにグループを追加する](#)
- [ユーザーアカウントの管理と検索](#)
- [パスワード、アカウント復旧、パスワードポリシー](#)
- [ユーザープールへのユーザーのインポート](#)
- [ユーザー属性の操作](#)

ユーザー作成ポリシーの設定

ユーザープールでは、ユーザーにサインアップを許可するか、ユーザーを管理者として作成できます。また、サインアップ後の検証と確認のプロセスを、どの程度ユーザーに委ねるかを制御できます。例えば、サインアップを確認して、外部の検証プロセスに基づいてサインアップを承認できます。この設定 (管理者作成ユーザーポリシー) では、ユーザーが自分のユーザーアカウントを確認できなくなるまでの時間も設定します。

Amazon Cognito は、ソフトウェアのカスタマー ID およびアクセス管理 (CIAM) プラットフォームとして、一般顧客のニーズに応えることができます。サインアップを受け入れてアプリケーションクライアントを持っているユーザープールは、マネージドログインの有無にかかわらず、パブリックに検出可能なアプリクライアント ID を知っていてサインアップをリクエストするインターネット上のすべてのユーザーに対して、ユーザープロフィールを作成します。サインアップしたユーザープロフィールは、アクセストークンと ID トークンを受け取り、アプリケーション用に承認されたリソースにアクセスできます。ユーザープールでサインアップを有効にする前に、オプションを確認し、設定がセキュリティ基準に準拠していることを確認してください。[自己登録を有効化] と `AllowAdminCreateUserOnly` を、以下の手順で説明するように注意して設定します。

AWS マネジメントコンソール

ユーザープールの [サインアップ] メニューには、ユーザープール内のユーザーのサインアップと管理者による作成に関する設定の一部が含まれています。

サインアップエクスペリエンスを設定するには

1. [Cognito による検証と確認] で、[Cognito が検証と確認のためのメッセージを自動的に送信することを許可] するかどうかを選択します。この設定を有効にすると、Amazon Cognito はユーザープールに提示する必要があるコードを含む E メールまたは SMS メッセージを新規ユーザーに送信します。これにより、E メールアドレスまたは電話番号の所有権が確認さ

れ、同等の属性が検証済みとして設定され、ユーザーアカウントのサインインが確認されます。選択した [検証する属性] により、検証メッセージの配信方法と送信先が決まります。

2. [属性変更の確認] は、ユーザーの作成時には重要ではありませんが、属性の検証に関係します。[\[サインイン属性\]](#) を変更したが、まだ検証していないユーザーに対して、新しい属性値または元の属性値を使用して引き続きサインインすることを許可できます。詳細については、「[ユーザーが E メールまたは電話番号を変更するときに確認する](#)」を参照してください。
3. [必須の属性] には、ユーザーのサインアップやユーザーの作成前に値を指定する必要がある属性が表示されます。必須属性は、ユーザープールの作成時にのみ設定できます。
4. カスタム属性は、ユーザーの作成とサインアップのプロセスにとって重要です。イミュータブルなカスタム属性の値を設定できるのは、ユーザーの初回作成時のみであるためです。カスタム属性の詳細については、「[カスタム属性](#)」を参照してください。
5. [認証されていない](#) SignUp API を使用して新しいアカウントを生成することをユーザーに許可する場合は、[セルフサービスのサインアップ] で、[自己登録を有効化] を設定します。自己登録を無効にした場合は、Amazon Cognito コンソールまたは [AdminCreateUser](#) API リクエストで新しいユーザーを管理者としてのみ作成できます。自己登録が非アクティブなユーザープールでは、[SignUp](#) API リクエストが `NotAuthorizedException` を返し、マネージドログインに [サインアップ] リンクが表示されません。

管理者としてユーザーを作成する予定のユーザープールでは、[認証方法] メニューの [管理者が設定した仮パスワードの有効期限] で仮パスワードの有効期間を設定できます。

管理者としてのユーザーを作成する場合の別の重要な要素は、招待メッセージです。新しいユーザーを作成すると、Amazon Cognito はアプリへのリンクを含むメッセージをユーザーに送信し、初回のサインインができるようにします。このメッセージテンプレートを [認証方法] メニューの [メッセージテンプレート] でカスタマイズします。

[\[機密アプリクライアント\]](#) (通常はウェブアプリケーション) には、アプリクライアントシークレットがないとサインアップできないようにするクライアントシークレットを設定できます。セキュリティ上のベストプラクティスとして、パブリックアプリクライアント (通常はモバイルアプリ) にはアプリクライアントシークレットを配布しないでください。クライアントシークレットを持つアプリケーションクライアントは、Amazon Cognito コンソールの [アプリケーションクライアント] メニューで作成できます。

Amazon Cognito user pools API

ユーザープールのユーザーを作成するためのパラメータは、[CreateUserPool](#) または [UpdateUserPool](#) API リクエストを使用してプログラムで設定できます。

[AdminCreateUserConfig](#) 要素は、ユーザープールの以下のプロパティの値を設定します。

1. セルフサービスのサインアップの有効化
2. 管理者が新規作成したユーザーに送信する招待メッセージ

次の例を API リクエスト本文全体に追加すると、セルフサービスのサインアップが無効なユーザープールと、基本的な招待メールが設定されます。

```
"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 {####}."
  }
}
```

[CreateUserPool](#) または [UpdateUserPool](#) API リクエストの以下の追加パラメータは、新規ユーザーの作成を制御します。

[AutoVerifiedAttributes](#)

新しいユーザーを登録するときに [自動的にメッセージを送信](#) する先の属性、メールアドレス、または電話番号。

[ポリシー](#)

ユーザープールの [パスワードポリシー](#)。

[Schema](#)

ユーザープールの [カスタム属性](#)。イミュータブルなカスタム属性の値を設定できるのはユーザーの初回作成時のみであるため、カスタム属性はユーザーの作成とサインアップのプロセスにとって重要です。

このパラメータでは、ユーザープールの必須の属性も設定します。次のテキストを API リクエスト本文全体の Schema 要素に挿入すると、email 属性が必要に応じて設定されます。

```
{
  "Name": "email",
```

```
    "Required": true  
  }
```

ユーザーアカウントのサインアップと確認

ユーザーアカウントは次の方法の 1 つでユーザープールに追加されます。

- ユーザーは、ユーザープールのクライアントアプリにサインアップします。これはモバイルまたはウェブアプリです。
- ユーザープールにユーザーのアカウントをインポートできます。詳細については、「[CSV ファイルからユーザープールへのユーザーのインポート](#)」を参照してください。
- ユーザープールにユーザーのアカウントを作成し、サインインへユーザーを招待することができす。詳細については、「[管理者としてのユーザーアカウントの作成](#)」を参照してください。

サインアップするユーザーは、サインインする前に確認が求められます。インポートされ作成されたユーザーはすでに確認されていますが、初めてサインインするときはパスワードを作成する必要があります。以下のセクションで、確認プロセス、E メール、電話確認について説明します。

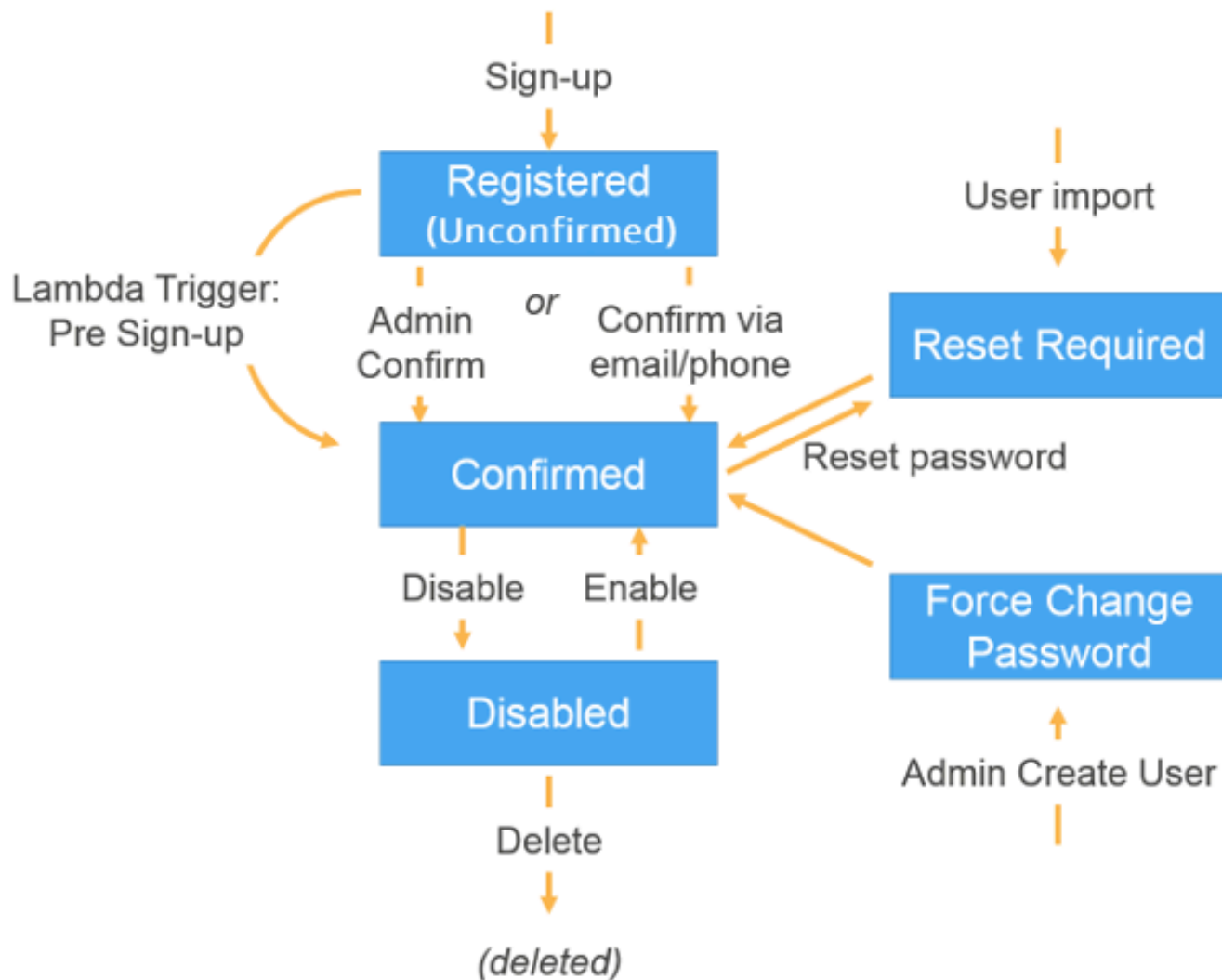
サインアップ時のパスワード

Amazon Cognito では、以下の条件を除き、サインアップ時にすべてのユーザーにパスワードの入力を求めます。以下の条件をすべて満たしている場合は、サインアップオペレーションでパスワードを省略できます。

1. [パスワードなしのサインイン](#) がユーザープールとアプリケーションクライアントでアクティブになっている。
2. アプリケーションは AWS SDK の認証モジュールを使用してカスタムビルドされます。マネージドログインとホストされた UI には、常にパスワードが必要です。
3. ユーザーが、許可されたパスワードなしのサインイン方法 (E メールまたは SMS メッセージのワンタイムパスワード (OTP) など) の属性値を指定している。例えば、E メールと電話の OTP によるサインインが許可されている場合、ユーザーは電話番号または E メールアドレスを指定できますが、E メールによるサインインのみが許可されている場合は、E メールアドレスを指定する必要があります。
4. ユーザープールが、パスワードなしのサインインでユーザーが使用できる属性を [自動的に検証](#) する。
5. どの [SignUp](#) リクエストでも、ユーザーは [Password](#) パラメータの値を指定しない。

ユーザーアカウント確認の概要

次の図は確認プロセスを示しています。



ユーザーアカウントは、以下のいずれかの状態になります。

登録済み (未確認)

ユーザーは正常にサインアップできますが、ユーザーアカウントを確認するまではサインインできません。ユーザーは有効ですが、この状態で確認されていません。

サインアップする新しいユーザーはこの状態から開始します。

確認済み

ユーザーアカウントが確認され、ユーザーはサインインできます。ユーザーがユーザーアカウントを確認するためにコードを入力したり、Eメールのリンクをたどったりすると、そのEメールや電話番号が自動的に認証されます。コードまたはリンクは24時間有効です。

ユーザーアカウントが管理者またはサインアップ前のLambdaトリガーによって確認された場合は、検証済みのEメールまたは電話番号がアカウントに関連付けられていないことがあります。

パスワードのリセットが必要

ユーザーアカウントは確認されましたが、ユーザーはサインインする前にコードをリクエストし、自身のパスワードをリセットする必要があります。

管理者またはデベロッパーがインポートしたユーザーアカウントは、この状態から開始します。

パスワードの強制変更

ユーザーアカウントが確認され、ユーザーは一時パスワードを使用してサインインできますが、初回のサインイン時に他の操作を行う前にパスワードを新しい値に変更する必要があります。

管理者またはデベロッパーが作成したユーザーアカウントは、この状態から開始します。

Disabled

ユーザーアカウントを削除する前に、そのユーザーのサインインアクセスを無効にする必要があります。

その他のリソース

- [Amazon Cognito による非アクティブなユーザーアカウントの検出と修正](#)

サインアップ時に連絡先情報を検証する

新しいユーザーがアプリにサインアップする際、1つ以上の連絡先の入力を求める必要があります。たとえば、ユーザーの連絡先情報は、以下のように使用されます。

- ユーザーが自分のパスワードをリセットする際、仮パスワードを送信する。
- ユーザーの個人情報または財務情報が更新された際に通知する。
- プロモーションメッセージ (特価販売や割引など) を送信する。

- アカウントの概要または請求日のお知らせを送信する。

このようなユースケースでは、検証済みの送信先にメッセージを送信することが重要です。そうしないと、誤入力された無効な E メールアドレスまたは電話番号にメッセージが送信される可能性があります。さらに悪いケースでは、ユーザーを騙る悪人に機密情報が送信される可能性があります。

適切なユーザーにのみメッセージが送信されることを確実にするには、サインアップ時にユーザーが以下の情報を入力することを必須とするように Amazon Cognito ユーザープールを設定します。

- a. E メールアドレスおよび電話番号。
- b. Amazon Cognito がその E メールアドレスまたは電話番号に送信する検証コード。24 時間が経過し、ユーザーのコードまたはリンクが有効でなくなった場合は、[ResendConfirmationCode](#) API オペレーションを呼び出して、新しいコードまたはリンクを生成して送信します。

検証コードを入力することで、コードを受信したメールボックスまたは電話へのアクセス権があるユーザーであることが証明されます。ユーザーがコードを入力すると、Amazon Cognito が以下を行うことによってユーザープールのユーザーに関する情報を更新します。

- ユーザーのステータスを CONFIRMED に設定する。
- ユーザーの属性を更新し、E メールアドレスまたは電話番号が検証されていることを示す。

この情報は、Amazon Cognito コンソールを使用して表示できます。または、AdminGetUser API オペレーション、で admin-get-user コマンド AWS CLI、またはいずれかの AWS SDKs で対応するアクションを使用できます。

ユーザーに検証済みの連絡方法がある場合は、ユーザーがパスワードリセットをリクエストする時に、Amazon Cognito が自動的にメッセージをユーザーに送信します。

ユーザー属性を確認および検証するその他のアクション

次のユーザーアクティビティでは、ユーザー属性を検証します。これらの属性を自動的に検証するように設定する必要はありません。リストされたアクションは、すべてのケースで属性を検証済みとしてマークします。

E メールアドレス

1. E メールワンタイムパスワード (OTP) を使用した[パスワードなしの認証](#)が正常に完了しました。
2. E メール OTP を使用した[多要素認証 \(MFA\)](#) が正常に完了しました。

Phone number (電話番号)

1. SMS OTP を使用した [パスワードなしの認証](#) が正常に完了しました。
2. SMS OTP を使用した [MFA](#) が正常に完了しました。

E メールまたは電話による検証を求めるようにユーザープールを設定するには

ユーザーの E メールアドレスと電話番号を確認することで、ユーザーに連絡できることを確認します。の次の手順を実行して AWS マネジメントコンソール、ユーザーが E メールアドレスまたは電話番号を確認するようにユーザープールを設定します。

Note

アカウントにまだユーザープールがない場合は、「[ユーザープールの開始方法](#)」を参照してください。

ユーザープールを設定するには

1. [Amazon Cognito コンソール](#) に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. ナビゲーションペインで、[User Pools] (ユーザープール) を選択します。リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [サインアップ] メニューを選択し、[属性の検証とユーザーアカウントの確認] を見つけます。[編集] を選択します。
4. [Cognito による検証と確認] で、[Cognito が検証と確認のためのメッセージを自動的に送信することを許可] するかどうかを選択します。この設定を有効にすると、Amazon Cognito は、ユーザーのサインアップ時またはユーザープロフィールの作成時に選択したユーザー連絡先属性にメッセージを送信します。サインイン用の属性を検証し、ユーザープロフィールを確認するために、Amazon Cognito はコードまたはリンクをメッセージでユーザーに送信します。次に、ユーザーは UI にコードを入力し、アプリが ConfirmSignUp または AdminConfirmSignUp API リクエストで確認できるようにする必要があります。

Note

[Cognito-assisted verification and confirmation] (Cognito アシストの検証と確認) を無効にして、API アクションまたは Lambda トリガーを使用して属性を検証し、ユーザーを確認することもできます。

このオプションを選択すると、Amazon Cognito はユーザーのサインアップ時に検証コードを送信しません。このオプションは、Amazon Cognito からの検証コードを使用せずに、少なくとも 1 つの連絡方法を検証するカスタムの認証フローを使用している場合に選択します。例えば、特定のドメインに属する E メールアドレスを自動的に確認するサインアップ前 Lambda トリガーを使用することができます。

ユーザーの連絡先情報を検証しない場合は、アプリケーションを使用できない場合があります。以下を行うには、ユーザーの連絡情報の検証が必要となる点にご注意ください。

- [Reset their passwords] (パスワードのリセット) — ユーザーが ForgotPassword API アクションを呼び出すオプションをアプリケーションで選択すると、Amazon Cognito はユーザーの E メールアドレスまたは電話番号に一時パスワードを送信します。Amazon Cognito がこのパスワードを送信するのは、ユーザーが検証済みの連絡方法を少なくとも 1 つ持っている場合のみです。
- [Sign in by using an email address or phone number as an alias] (E メールアドレスまたは電話番号をエイリアスとして使用してサインインする) — これらのエイリアスを許可するようにユーザープールを設定した場合、ユーザーは、検証済みである場合にのみ、そのエイリアスを使用してサインインできます。詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

5. [Attributes to verify] (検証する属性) を以下から選択します。

SMS メッセージを送信し、電話番号を確認する

ユーザーがサインアップすると Amazon Cognito が SMS メッセージで検証コードを送信します。通常 SMS メッセージでユーザーと通信する場合は、このオプションを選択してください。例えば、配信通知、予約確認、または警告を送信する場合は、検証済みの電話番号を使用します。アカウントが確認されると、ユーザーの電話番号が検証済み属性になります。ユーザーの E メールアドレスを確認して通信するには、追加のアクションを実行する必要があります。

E メールメッセージを送信し、E メールアドレスを確認する

ユーザーがサインアップすると Amazon Cognito が E メール経由で検証コードを送信します。通常 E メールでユーザーと通信する場合は、このオプションを選択してください。たとえば、請求明細、注文書、特価販売を使用する場合は、検証済みの E メールアドレスを使用する必要があります。アカウントが確認されると、ユーザーのメールアドレスが検証済み属

性になります。ユーザーの電話番号を確認して通信するには、追加のアクションを実行する必要があります。

電話番号が利用可能な場合は SMS メッセージを送信し、そうでない場合は E メールメッセージを送信する

すべてのユーザーに同じ検証済みの連絡方法を要求する必要がない場合は、このオプションを選択してください。この場合、アプリのサインアップページで、希望する連絡方法のみを検証するようにユーザーに求めることができます。Amazon Cognito が検証コードを送信するときは、アプリからの SignUp リクエストで指定された連絡方法にコードが送信されます。ユーザーが E メールアドレスと電話番号の両方を入力し、アプリが SignUp リクエストに両方の連絡方法を指定する場合、Amazon Cognito は検証コードを電話番号のみに送信します。

ユーザーが E メールアドレスと電話番号の両方を検証することを必須としている場合は、このオプションを選択してください。Amazon Cognito は、ユーザーのサインアップ時に 1 つの連絡方法を検証するので、アプリがユーザーのサインイン後にもう一方の連絡方法を検証する必要があります。詳細については、「[ユーザーに E メールアドレスと電話番号の両方の確認を要求する場合](#)」を参照してください。

6. [Save changes] (変更の保存) をクリックします。

E メールアドレスまたは電話による検証を使用した認証フロー

ユーザープールで連絡先情報の検証をユーザーに要求する場合、ユーザーがアプリにサインアップした後のフローを容易にする必要があります。

1. ユーザーは、ユーザー名、メールアドレス、電話番号やその他の属性を入力してアプリにサインアップします。
2. Amazon Cognito サービスは、アプリからサインアップリクエストを受け取ります。サインアップに必要なすべての属性がリクエストに含まれることを確認した後で、サービスはサインアッププロセスを完了し、確認コードをユーザーの電話 (SMS メッセージ) または E メールに送信します。コードは 24 時間有効です。
3. サービスは、サインアップが完了しユーザーアカウントが確認保留であることを返します。応答には、確認コードの送信先に関する情報が含まれます。この時点でユーザーアカウントは未確認状態になり、ユーザーのメールアドレスと電話番号は未確認です。
4. アプリは、ユーザーに確認コードを入力するよう求めるメッセージを表示できます。ユーザーは、コードをすぐに入力する必要はありません。ただし、確認コードを入力するまでユーザーはサインインできません。

5. ユーザーがアプリに確認コードを入力します。
6. アプリは、Amazon Cognito サービスにコードを送信する [ConfirmSignUp](#) を呼び出します。これは、コードを検証し、コードが正しい場合はユーザーのアカウントを確認済み状態に設定します。ユーザーアカウントが正常に確認されると、Amazon Cognito サービスが自動的に、E メールアドレスまたは電話番号を確認するために使用された属性を検証済みとしてマークします。この属性の値が変更されていない場合、ユーザーはそれを再度確認する必要はありません。
7. この時点で、ユーザーアカウントは確認済みの状態になっており、ユーザーはサインインできます。

ユーザーに E メールアドレスと電話番号の両方の確認を要求する場合

Amazon Cognito は、ユーザーのサインアップ時に 1 つの連絡方法しか検証しません。Amazon Cognito が E メールアドレスまたは電話番号のどちらを検証するかを選択する必要がある場合は、SMS メッセージ経由で検証コードを送信して電話番号を検証することを選択します。例えば、ユーザーが E メールアドレスと電話番号のいずれかを検証できるようにユーザープールを設定する場合、およびアプリがサインアップ時にこれら両方の属性を提供する場合、Amazon Cognito は電話番号のみを検証します。ユーザーが電話番号を検証したら、Amazon Cognito がユーザーのステータスを CONFIRMED に設定し、ユーザーはアプリにサインインできるようになります。

ユーザーがアプリにサインインした後、サインアップ時に検証されなかった連絡方法を検証するオプションを提示することができます。この 2 つ目の方法を検証するには、アプリで `VerifyUserAttribute` API アクションを呼び出します。このアクションには `AccessToken` パラメータが必要で、Amazon Cognito は認証されたユーザーにアクセストークンしか提供しないことに注意してください。したがって、2 つ目の連絡方法は、ユーザーがサインインした後にのみ検証できます。

E メールアドレスと電話番号の両方を検証するようにユーザーに求める場合は、以下のように行います。

1. E メールアドレスまたは電話番号を検証することをユーザーに許可するようにユーザープールを設定します。
2. アプリのサインアップフローで、E メールアドレスおよび電話番号の両方を入力するようユーザーに求めます。 [SignUp](#) API アクションを呼び出し、`UserAttributes` パラメータの E メールアドレスおよび電話番号を指定します。この時点で、Amazon Cognito がユーザーの携帯電話に検証コードを送信します。
3. アプリのインターフェイスで、ユーザーが検証コードを入力する確認ページを提示します。ユーザーを確認するには、 [ConfirmSignUp](#) API アクションを呼び出します。この時点で、ユーザー

のステータスは CONFIRMED となり、ユーザーの電話番号は検証されますが、E メールアドレスはまだ検証されていません。

4. サインインページを提示し、ユーザーを認証するには、[InitiateAuth](#) API アクションを呼び出します。ユーザーが認証されると、Amazon Cognito がアプリにアクセストークンを返します。
5. [GetUserAttributeVerificationCode](#) API アクションを呼び出します。リクエストで次のパラメータを指定します。
 - AccessToken – ユーザーのサインイン時に Amazon Cognito が返したアクセストークン。
 - AttributeName – "email" を属性値として指定します。

Amazon Cognito がユーザーの E メールアドレスに検証コードを送信します。

6. ユーザーが検証コードを入力する確認ページを提示します。ユーザーがコードを入力したら、[VerifyUserAttribute](#) API アクションを呼び出します。リクエストで次のパラメータを指定します。
 - AccessToken – ユーザーのサインイン時に Amazon Cognito が返したアクセストークン。
 - AttributeName – "email" を属性値として指定します。
 - Code – ユーザーが提供した検証コード。

この時点で、E メールアドレスが検証されます。

ユーザーにアプリケーションへのサインアップを許可するがユーザープール管理者として確認する

ユーザープールで確認メッセージを自動的に送信しないにも関わらず、誰でもアカウントにサインアップできるようにしたい場合があります。このモデルでは、たとえば、新しい登録リクエストを人間が確認したり、サインアップの一括検証と処理を行ったりする余地があります。Amazon Cognito コンソール、または IAM 認証された API オペレーション [AdminConfirmSignUp](#) を使用して新しいユーザーアカウントを確認できます。ユーザープールが確認メッセージを送信するかどうかにかかわらず、ユーザーアカウントを管理者として確認できます。

この方法では、ユーザーのセルフサービスサインアップを確認することしかできません。管理者として作成したユーザーを確認するには、Permanent を True に設定した [AdminSetUserPassword](#) API リクエストを作成します。

1. ユーザーは、ユーザー名、メールアドレス、電話番号やその他の属性を入力してアプリにサインアップします。
2. Amazon Cognito サービスは、アプリからサインアップリクエストを受け取ります。サインアップに必要なすべての属性がリクエストに含まれることを確認した後で、サービスはサインアッププロセスを完了し、アプリにサインアップが完了し確認が保留中であることをアプリに返します。この時点でユーザーのアカウントは未確認状態です。アカウントが確認されるまで、ユーザーはサインインすることはできません。
3. ユーザーのアカウントを確認します。アカウントを確認するには、 にサインイン AWS マネジメントコンソール するか、 AWS 認証情報を使用して API リクエストに署名する必要があります。
 - a. Amazon Cognito コンソールでユーザーを確認するには、[ユーザー] メニューに移動して、確認するユーザーを選択し、[アクション] メニューの [確認] を選択します。
 - b. AWS API または CLI でユーザーを確認するには、 [AdminConfirmSignUp](#) API リクエストを作成するか、 で [admin-confirm-sign-up](#) を作成します AWS CLI。
4. この時点で、ユーザーアカウントは確認済みの状態になっており、ユーザーはサインインできます。

シークレットハッシュ 値の計算

ベストプラクティスとして、機密アプリケーションクライアントにクライアントシークレットを割り当てます。アプリケーションクライアントにクライアントシークレットを割り当てる場合、Amazon Cognito ユーザープール API リクエストには、リクエスト本文にクライアントシークレットを含むハッシュを含める必要があります。以下のリストにある API オペレーションのクライアントシークレットに関する知識を検証するには、クライアントシークレットをアプリケーションのクライアント ID とユーザーのユーザー名と連結し、その文字列を base64 でエンコードします。

シークレットハッシュを持つクライアントにアプリがユーザーをサインインすると、ユーザープールのサインイン属性の値をシークレットハッシュのユーザー名要素として使用できます。アプリが REFRESH_TOKEN_AUTH による認証オペレーションで新しいトークンをリクエストする場合、ユーザー名要素の値はサインイン属性によって異なります。ユーザープールにサインイン属性として username がない場合は、ユーザーのアクセストークンまたは ID トークンの sub クレームからのシークレットハッシュユーザー名の値を設定します。username がサインイン属性である場合は、username クレームからのシークレットハッシュユーザー名の値を設定します。

以下の Amazon Cognito ユーザープール API は、SecretHash パラメータにクライアントシークレットのハッシュ値を受け入れます。

- [ConfirmForgotPassword](#)
- [ConfirmSignUp](#)
- [ForgotPassword](#)
- [ResendConfirmationCode](#)
- [SignUp](#)

また、以下の API は、認証パラメータまたはチャレンジレスポンス内の SECRET_HASH パラメータにクライアントシークレットのハッシュ値を受け入れます。

API オペレーション:	SECRET_HASH の親パラメータ
InitiateAuth	AuthParameters
AdminInitiateAuth	AuthParameters
RespondToAuthChallenge	ChallengeResponses
AdminRespondToAuthChallenge	ChallengeResponses

シークレットハッシュ値は、Base 64 でエンコードされたキー付きハッシュメッセージ認証コード (HMAC) であり、ユーザープールクライアントおよびユーザー名、さらにメッセージ内のクライアント ID を使用して計算されたものです。次の擬似コードは、この値の計算方法を示します。この擬似コードで + は連結を表し、HMAC_SHA256 は HmacSHA256 を使用して HMAC 値を生成する関数を、Base64 は Base-64 でエンコードされたバージョンのハッシュ出力を生成する関数を示します。

```
Base64 ( HMAC_SHA256 ( "Client Secret Key", "Username" + "Client Id" ) )
```

SecretHash パラメータの計算方法と使用方法の詳細な概要については、AWS ナレッジセンターの[Amazon Cognito ユーザープール API からのクライアント <client-id> エラーのシークレットハッシュを検証できない](#)のトラブルシューティング方法を教えてください。

サーバー側の Java アプリケーションコードで、次のコード例を使用できます。

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)
```

E メールまたは電話番号の確認なしでユーザーアカウントを確認する

検証コードを要求したり、E メールまたは電話番号を検証したりすることなくサインアップ時にユーザーアカウントを自動的に確認するには、サインアップ前の Lambda トリガーを使用できます。この方法で確認されたユーザーは、コードを受け取らないですぐにサインインできます。

このトリガーによって検証されたユーザーのメールまたは電話番号をマーキングすることもできます。

Note

これはユーザーが始めるにあたって便利な方法ですが、E メールまたは電話番号を少なくとも 1 つ自動検証することをお勧めします。そうしないと、ユーザーがパスワードを忘れた際に復旧できないままになる場合があります。

ユーザーがサインアップ時に検証コードを受信して入力することを必須とせず、サインアップ前の Lambda トリガーで E メールと電話番号を自動検証しない場合は、そのユーザーアカウントに検証済みの E メールアドレスまたは電話番号が存在しないリスクが生じます。ユーザーはメールアドレスまたは電話番号を後で確認できます。ただし、パスワードを忘れて確認済みの E メールアドレスまたは電話番号がない場合、パスワードを忘れた場合のフローで検証コードをユーザーに送信するために確認済みの E メールまたは電話番号が必要となるため、ユーザーはアカウントからロックアウトされます。

ユーザーが E メールまたは電話番号を変更するときに確認する

複数のサインイン名で設定したユーザープールでは、ユーザーはサインイン時にユーザー名として電話番号または E メールアドレスを入力できます。アプリケーションで E メールアドレスまたは電話番号を更新すると、Amazon Cognito は新しい属性値の所有権を検証するコードを含むメッセージをすぐに送信できます。これらの検証コードの自動送信を有効にするには、「[E メールまたは電話による検証の設定](#)」を参照してください。

検証コードを受け取ったユーザーは、[VerifyUserAttribute](#) リクエストでそのコードを Amazon Cognito に返す必要があります。コードを返すと、その属性が検証済みとしてマークされます。通常、ユーザーが E メールアドレスまたは電話番号を更新した場合、更新した値を使用してサインインしてメッセージを受信する前に、ユーザーが更新した値の所有者であることを確認する必要があります。ユーザープールには、ユーザーが E メールアドレスや電話番号の更新を確認する必要があるかどうかを決定する設定可能なオプションがあります。

このオプションは、ユーザープールのプロパティ

`AttributesRequireVerificationBeforeUpdate` です。これを設定するには、[CreateUserPool](#) または [UpdateUserPool](#) リクエストを使用するか、Amazon Cognito コンソールの [サインアップ] メニューで [更新が保留中のときに元の属性値をアクティブのままにする] を使用します。

ユーザープールが E メールアドレスと電話番号の更新を処理する方法は、ユーザープールのユーザー名設定に関連しています。ユーザープールのユーザー名は、[ユーザー名属性] 設定に含めることができます。この設定では、E メールアドレス、電話番号、またはその両方がサインイン名となります。また、[エイリアス属性] 設定に含めることもできます。この設定では、username 属性がサインイン名となり、E メールアドレス、電話番号、または希望するユーザー名が代替サインイン名となります。詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

カスタムメッセージの Lambda トリガーを使用して確認メッセージをカスタマイズすることもできます。詳細については、「[カスタムメッセージの Lambda トリガー](#)」を参照してください。ユーザーの E メールアドレスや電話番号が未確認の場合、アプリケーションはユーザーに属性を確認する必要があることを通知し、新しい E メールアドレスや電話番号を確認するためのボタンやリンクを提供する必要があります。

次の表は、ユーザーがサインイン属性の値を変更したときに、`AttributesRequireVerificationBeforeUpdate` およびエイリアス設定でどのように結果を決定するかを示しています。

ユーザー名の設定	ユーザーが新しい属性を確認する必要がある場合の動作	ユーザーが新しい属性を確認する必要がない場合の動作
ユーザー名属性	元の属性は、検証済みで、サインイン可能であり、元の値のままです。ユーザーが新しい値を確認すると、Amazon Cognito は属性値を更新し、検証済みとしてマークし、サインイン可能にします。	Amazon Cognito は属性を新しい値に更新します。新しい値でサインイン可能になります。ユーザーが新しい値を確認すると、Amazon Cognito はそれを検証済みとしてマークします。
エイリアス属性	元の属性は、検証済みで、サインイン可能であり、元の値のままです。ユーザーが新しい値を確認すると、Amazon Cognito は属性値を更新し、	Amazon Cognito は属性を新しい値に更新します。元の属性値でも新しい属性値でもサインインできません。ユーザーが新しい値を確認する

ユーザー名の設定	ユーザーが新しい属性を確認する必要がある場合の動作	ユーザーが新しい属性を確認する必要がない場合の動作
	検証済みとしてマークし、サインイン可能にします。	と、Amazon Cognito は属性値を更新し、検証済みとしてマークし、サインイン可能にします。

例 1

ユーザー 1 は、E メールアドレス `user1@example.com` でアプリケーションにサインインし、ユーザー名 `user1` (エイリアス属性) を持っています。ユーザープールは、サインイン属性の更新を検証し、検証メッセージを自動的に送信するように設定されています。ユーザーは E メールアドレスを `user1+foo@example.com` に更新するようリクエストします。ユーザーは、確認 E メールを `user1+foo@example.com` で受け取り、E メールアドレス `user1@example.com` のみを使用して再度サインインできます。その後、確認コードを入力し、E メールアドレス `user1+foo@example.com` のみを使用して再度サインインできます。

例 2

ユーザー 2 は、E メールアドレス `user2@example.com` を使用してアプリケーションにサインインし、ユーザー名 (エイリアス属性) を持っています。ユーザープールは、サインイン属性の更新を検証したり、検証メッセージを自動的に送信したりしないように設定されています。ユーザーは E メールアドレスを `user2+bar@example.com` に更新するようリクエストします。ユーザーは `user2+bar@example.com` で確認 E メールを受け取り、再度サインインすることはできません。その後、確認コードを入力し、E メールアドレス `user2+bar@example.com` のみを使用して再度サインインできます。

例 3

ユーザー 3 は、E メールアドレス `user3@example.com` を使用してアプリケーションにサインインします。ユーザー名 (ユーザー名属性) は持っていません。ユーザープールは、サインイン属性の更新を検証したり、検証メッセージを自動的に送信したりしないように設定されています。ユーザーは E メールアドレスを `user3+baz@example.com` に更新するようリクエストします。ユーザーは確認 E メールを `user3+baz@example.com` で受け取りますが、検証コードで追加のアクションを実行することなく、即座にサインインできます。

管理者またはデベロッパーが作成するユーザーアカウントの確認および検証プロセス

管理者またはデベロッパーが作成したユーザーアカウントは常に確認済み状態にあるため、ユーザーは確認コードを入力する必要はありません。Amazon Cognito サービスがこれらのユーザーに送信する招待メッセージには、ユーザー名と一時パスワードが含まれています。ユーザーは、サインインする前にパスワードを変更する必要があります。詳細については、は、「[E メールメッセージと SMS メッセージのカスタマイズ](#)」の「[管理者としてのユーザーアカウントの作成](#)」と「[Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#)」のカスタムメッセージのトリガーを参照してください。

インポート済みユーザーアカウントの確認および検証プロセス

、CLI AWS マネジメントコンソール、または API のユーザーインポート機能を使用して作成されたユーザーアカウント（「」を参照[CSV ファイルからユーザープールへのユーザーのインポート](#)）は既に確認済み状態であるため、ユーザーは確認コードを入力する必要はありません。招待メッセージは送信されません。ただし、インポートされたユーザーアカウントでは、サインインする前にまず ForgotPassword API を呼び出してコードをリクエストし、次に、提供されたコードを使用して ConfirmForgotPassword API を呼び出すことによってパスワードを作成する必要があります。詳細については、「[インポートされたユーザーにパスワードをリセットするように要求](#)」を参照してください。

ユーザーアカウントがインポートされたときに、ユーザーの E メールまたは電話番号は確認済みとしてマークする必要があります。したがって、ユーザーがサインインする際に認証は必要ありません。

アプリケーションのテスト中に E メールを送信する

ユーザープールのクライアントアプリでアカウントを作成および管理する場合は、Amazon Cognito がユーザーに E メールメッセージを送信します。E メール検証を必須とするようにユーザープールを設定した場合は、以下のタイミングで Amazon Cognito が E メールを送信します。

- ユーザーのサインアップ時。
- ユーザーが E メールアドレスを更新する際。
- ユーザーが ForgotPassword API アクションを呼び出すアクションを実行する際。
- 管理者としてユーザーアカウントを作成する際。

E メールを開始するアクションに応じて、検証コードまたは仮パスワードが含まれます。ユーザーは、これらの E メールを受け取り、メッセージを理解する必要があります。そうしないと、アプリにサインインして使用できない場合があります。

E メールが正常に送信され、メッセージの内容が正しいことを確認するには、Amazon Cognito からの E メール配信を開始するアプリのアクションをテストします。たとえば、アプリでサインアップページを使用するか、SignUp API アクションを使用して、テスト用メールアドレスでサインアップすることで E メールを配信できます。この方法でテストする場合は、次の点に注意してください。

i [重要]

Amazon Cognito からの E メールを開始するアクションをテストするために E メールアドレスを使用する場合は、偽の E メールアドレス (メールボックスが存在しないメールアドレス) は使用しないでください。ハードバウンスを発生させずに Amazon Cognito からの E メールを受信する実際の E メールアドレスを使用します。

ハードバウンスは、Amazon Cognito が受取人のメールボックスへの Eメールの配信に失敗した場合に発生します。これは、常にメールボックスが存在しない場合に発生します。

Amazon Cognito は、永続的にハードバウンスが発生する AWS アカウントによって送信できる Eメールの数を制限します。

E メールを配信するアクションをテストする場合、次のいずれかの E メールアドレスを使用してハードバウンスを回避します。

- テスト用に所有し、使用している E メールアカウントのメールアドレス。独自の E メールアドレスを使用すると、Amazon Cognito が送信する E メールを受け取ります。この E メールでは、検証コードを使用して、アプリのサインアップエクスペリエンスをテストできます。ユーザープールの E メールメッセージをカスタマイズした場合は、正しく表示されていることを確認します。
- メールボックスシミュレーターのアドレス (success@simulator.amazonses.com)。シミュレーターのアドレスを使用する場合、Amazon Cognito は E メールを正常に送信しますが、それを表示することはできません。このオプションは、検証コードを使用する必要がなく、Eメールのメッセージを確認する必要がない場合に役立ちます。
- success+user1@simulator.amazonses.com または success+user2@simulator.amazonses.com などの、任意のラベルを使用するメールボックスシミュレーターのアドレス。Amazon Cognito はこれらのアドレスに E メールを正常に送信しますが、送信された E メールを表示することはできません。このオプションは、複数のテストユーザーをユーザープールに追加してサインアッププロセスをテストし、各テストユーザーに一意の E メールアドレスがある場合に便利です。

E メールまたは電話による検証の設定

E メールまたは電話の検証の設定は、[認証方法] メニューで選択できます。多要素認証 (MFA) の詳細については、「[SMS テキストメッセージ MFA](#)」を参照してください。

Amazon Cognito は、SMS メッセージの送信に Amazon SNS を使用します。Amazon Cognito または他の から SMS メッセージを送信したことがない場合 AWS のサービス、Amazon SNS はアカウントを SMS サンドボックスに配置することがあります。アカウントをサンドボックスから本番環境に削除する前に、検証済みの電話番号にテストメッセージを送信することをお勧めします。さらに、米国の宛先電話番号に SMS メッセージを送信する場合は、Amazon Pinpoint から発信元 ID または送信者 ID を取得する必要があります。SMS メッセージ用に Amazon Cognito ユーザープールを設定するには、「[Amazon Cognito ユーザープール用の SMS メッセージ設定](#)」を参照してください。

Amazon Cognito は、E メールアドレスまたは電話番号を自動的に検証できます。この検証を行うために、Amazon Cognito は検証コードまたは検証リンクを送信します。E メールアドレスの場合、Amazon Cognito は、コードまたはリンクを E メールメッセージで送信できます。Amazon Cognito コンソールの [メッセージテンプレート] メニューで [検証メッセージ] テンプレートを編集するときに、[検証タイプ] として [コード] または [リンク] を選択できます。詳細については、「[E メール検証メッセージのカスタマイズ](#)」を参照してください。

電話番号の場合、Amazon Cognito は、コードを SMS テキストメッセージで送信します。

Amazon Cognito は、ユーザーを確認し、パスワードを忘れた場合の回復を支援するために、電話番号または E メールアドレスを検証する必要があります。または、サインアップ前の Lambda トリガーまたは [AdminConfirmSignUp](#) API オペレーションを使用することで、ユーザーを自動的に確認できます。詳細については、「[ユーザーアカウントのサインアップと確認](#)」を参照してください。

確認コードまたはリンクは 24 時間有効です。

E メールアドレスや電話番号の検証を必須とした場合、Amazon Cognito はユーザーのサインアップ時に自動的に検証コードまたはリンクを送信します。ユーザープールに [カスタム SMS 送信者の Lambda トリガー](#) または [カスタム E メール送信者の Lambda トリガー](#) を設定すると、その関数が代わりに呼び出されます。

注意事項

- 電話番号の検証に SMS テキストメッセージングを使用すると、Amazon SNS の料金が別途請求されます。E メールメッセージを送信しても料金はかかりません。Amazon SNS の料金については、「[Worldwide SMS Pricing](#)」を参照してください。SMS メッセージを利

用可能な国の最新のリストについては、「[サポートされるリージョンと国](#)」を参照してください。

- Amazon Cognito からの E メールメッセージを生成するアプリ内のアクションをテストするときは、Amazon Cognito がハードバウンスを発生させずに送信できる実際の E メールアドレスを使用してください。詳細については、「[the section called “アプリケーションのテスト中に E メールを送信する”](#)」を参照してください。
- パスワードを忘れた場合のフローでは、ユーザーのメールまたはユーザーの電話番号が検証される必要があります。

Important

ユーザーが電話番号と E メールアドレスの両方にサインアップし、ユーザープール設定で両方の属性の確認が必要な場合は、Amazon Cognito は、検証コードを SMS メッセージ経由で電話番号に送信します。Amazon Cognito はまだ E メールアドレスを検証していないため、アプリは [GetUser](#) を呼び出して、E メールアドレスが検証待ちかどうかを確認する必要があります。検証が必要な場合、アプリは、[GetUserAttributeVerificationCode](#) を呼び出し、E メール検証フローを開始する必要があります。その後、[VerifyUserAttribute](#) を呼び出して検証コードを送信する必要があります。

AWS アカウント および個々のメッセージの SMS メッセージの支出クォータを調整できます。制限は SMS メッセージを送信するコストにのみ適用されます。詳細については、[Amazon SNS よくある質問](#)の「アカウントレベルとメッセージレベルの支出クォータとは何ですか。また、どのように機能しますか。」を参照してください。

Amazon Cognito は、ユーザープールを作成した AWS リージョン または次の表のレガシー Amazon SNS 代替リージョンのいずれかで、Amazon SNS リソースを使用して SMS メッセージを送信します。Amazon SNS 例外は、アジアパシフィック (ソウル) リージョンの Amazon Cognito ユーザープールです。これらのユーザープールは、アジアパシフィック (東京) リージョンで Amazon SNS 設定を使用します。詳細については、「[SMS メッセージの AWS リージョン を選択する](#)」を参照してください。

Amazon Cognito リージョン	レガシー Amazon SNS 代替リージョン
米国東部(オハイオ)	米国東部 (バージニア北部)

Amazon Cognito リージョン	レガシー Amazon SNS 代替リージョン
アジアパシフィック (ムンバイ)	アジアパシフィック (シンガポール)
アジアパシフィック (ソウル)	アジアパシフィック (東京)
カナダ (中部)	米国東部 (バージニア北部)
欧州 (フランクフルト)	欧州 (アイルランド)
欧州 (ロンドン)	欧州 (アイルランド)

例: Amazon Cognito のユーザープールがアジアパシフィック (ムンバイ) にあり、ap-southeast-1 で使用制限を引き上げている場合は、ap-south-1 で別途引き上げる要求をしないでください。代わりに、アジアパシフィック (シンガポール) で Amazon SNS リソースを使用できます。

E メールアドレスと電話番号の更新を検証する

E メールアドレスまたは電話番号の属性は、ユーザーが値を変更した直後にアクティブになり、検証されない場合があります。Amazon Cognito は、Amazon Cognito が属性を更新する前に、ユーザーに新しい値を検証するように要求することもできます。ユーザーが新しい値を最初に検証することを要求する場合、新しい値を検証するまでは、サインインおよびメッセージの受信に元の値を使用することができます。

ユーザーがユーザープールの E メールアドレスまたは電話番号をサインインエイリアスとして使用できる場合、更新された属性のサインイン名は、更新された属性の検証が必要かどうかによって異なります。更新された属性を検証する必要がある場合、新しい値を検証するまで、ユーザーは元の属性値を使用してサインインできます。更新された属性を検証する必要がある場合、新しい値を検証するまで、ユーザーは新しい属性値または元の属性値でサインインまたはメッセージを受信することはできません。

例えば、ユーザープールでは、E メールアドレスのエイリアスでのサインインが許可され、ユーザーが更新時に E メールアドレスを検証する必要があるとします。sue@example.com としてサインインしている Sue は、自分の E メールアドレスを sue2@example.com に変更しようとしたが、誤って ssue2@example.com と入力してしまいました。Sue は確認用の E メールを受信していないので、ssue2@example.com を検証することができません。sue@example.com としてサインインし、アプリでフォームを再送信して、E メールアドレスを sue2@example.com に更新します。この E メールを受信し、アプリに確認コードを提供して、sue2@example.com としてサインインを開始します。

ユーザーが属性を更新し、ユーザープールが新しい属性値を検証する場合

- コードを確認する前は、元の属性値でサインインして新しい値を確認できます。
- コードを確認した後は、新規の属性値でサインインして新しい値を確認できます。
- [AdminUpdateUserAttributes](#) API リクエストで `email_verified` または `phone_number_verified` を `true` に設定すると、Amazon Cognito から送信されたコードを確認する前にサインインできるようになります。

ユーザーが属性を更新し、ユーザープールが新しい属性値を検証しない場合

- 元の属性値を使用してサインインしたり、元の属性値でメッセージを受信したりすることはできません。
- 新しい値を確認するコードを確認するま、新しい属性値を使用してサインインしたり、確認コード以外のメッセージを受信したりすることはできません。
- [AdminUpdateUserAttributes](#) API リクエストで `email_verified` または `phone_number_verified` を `true` に設定すると、Amazon Cognito から送信されたコードを確認する前にサインインできるようになります。

ユーザーが E メールアドレスまたは電話番号を更新するときに属性の検証を要求するには

1. [Amazon Cognito コンソール](#) にサインインします。プロンプトが表示されたら、AWS 認証情報を入力します。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [サインアップ] メニューで、[属性の検証とユーザーアカウントの確認] の [編集] を選択します。
4. [Keep original attribute value active when an update is pending] (更新が保留中の場合、元の属性値をアクティブに保つ) を選択します。
5. [Active attribute values when an update is pending] (更新が保留中の場合にアクティブな属性値) から、Amazon Cognito が値を更新する前にユーザーに検証を要求する属性を選択します。
6. [Save changes] (変更の保存) をクリックします。

Amazon Cognito API で属性更新の検証を要求するには、[UpdateUserPool](#) リクエストで `AttributesRequireVerificationBeforeUpdate` パラメータを設定します。

SMS メッセージを代理送信するために Amazon Cognito を承認します。

SMS メッセージをユーザーに代理送信するには、Amazon Cognito に許可が必要です。そのアクセス許可を付与するには、AWS Identity and Access Management (IAM) ロールを作成します。Amazon Cognito コンソールの [認証方法] メニューの [SMS] で、[編集] を選択してロールを設定します。

MFA、認証、検証、招待メッセージの設定

Amazon Cognito を使用すると、SMS や E メール認証、検証、およびユーザー招待メッセージをカスタマイズして、アプリケーションのセキュリティとユーザーエクスペリエンスを強化できます。一部のメッセージでは、コードベースの検証またはワンクリックリンクによる検証を選択できます。このトピックでは、Amazon Cognito コンソールで認証と検証の通信をパーソナライズする方法について説明します。

[メッセージテンプレート] メニューで、以下をカスタマイズできます。

- ワンタイムパスワード (OTP) と多要素認証 (MFA) 用の E メールおよび SMS メッセージテンプレート
- SMS と Eメールの検証メッセージ
- Eメールの検証タイプ (コードまたはリンク)

Note

Amazon Cognito は、ユーザーがサインアップまたは確認コードを再送信するときに、リンクとリンクベースのテンプレートを含む検証メッセージを送信します。コードテンプレートは、属性の更新オペレーションとパスワードのリセットオペレーションに関連する E メールで使用します。

- ユーザーの招待メッセージ
- ユーザープールを介するメールの FROM および REPLY-TO E メールアドレス

Note

SMS と Eメールの検証メッセージテンプレートは、電話番号と Eメールの検証を必須にした場合にのみ表示されます。同様に、SMS MFA メッセージのテンプレートは、MFA 設定が [required] (必要) または [optional] (任意) の場合にのみ表示されます。

トピック

- [メッセージテンプレート](#)
- [E メールと SMS の MFA メッセージのカスタマイズ](#)
- [E メール検証メッセージのカスタマイズ](#)
- [ユーザー招待メッセージのカスタマイズ](#)
- [E メールアドレスのカスタマイズ](#)
- [Amazon SES E メールを代理送信するための Amazon Cognito の承認 \(カスタム REPLY-TO E メールアドレスからの送信\)](#)

メッセージテンプレート

メッセージテンプレートを使用して、メッセージにプレースホルダーを挿入できます。Amazon Cognito はプレースホルダーを、対応する値に置き換えます。これらの値はすべてのメッセージタイプに存在するわけではありませんが、任意のタイプのメッセージテンプレートでユニバーサルテンプレートプレースホルダーを参照できます。

ユニバーサルテンプレートプレースホルダー

説明	トークン	メッセージタイプ
検証コード	{#####}	検証、確認、MFA メッセージ
一時パスワード	{#####}	パスワードを忘れた場合と招待メッセージ
ユーザー名	{username}	招待メッセージと高度なセキュリティメッセージ

[脅威保護](#)で利用可能な自動レスポンスの 1 つは、Amazon Cognito が潜在的に悪意のあるアクティビティを検出したことをユーザーに知らせることです。アドバンスドセキュリティのテンプレートのプレースホルダーを使用して、以下を行うことができます。

- IP アドレス、市、国、サインイン時間、デバイス名など、イベントに関する具体的な詳細を記載します。Amazon Cognito の脅威保護は、これらの詳細を分析できます。
- ワンクリックリンクが有効であるかどうかを確認します。

- イベント ID、フィードバックトークン、およびユーザー名を使用して独自のワンクリックリンクを構築します。

Note

ワンクリックリンクを生成し、アドバンスドセキュリティ E メールテンプレートで {one-click-link-valid} および {one-click-link-invalid} プレースホルダーを使用するには、ユーザープール用にドメインが既に設定されている必要があります。

脅威保護では、以下のプレースホルダーを追加してメッセージテンプレートに挿入できるようにします。これらのプレースホルダーは、アダプティブ認証メッセージに適用されます。このメッセージは、ユーザーのセッションがリスクレベルの評価を受けた場合に、Amazon Cognito からユーザーに送信する通知です。これらの変数を使用してメッセージテンプレートを設定するには、Amazon Cognito コンソールで脅威保護の [フル機能] 設定を更新するか、[SetRiskConfiguration](#) リクエストでテンプレートを送信します。

アドバンスドセキュリティのテンプレートのプレースホルダー

説明	トークン
IP アドレス	{ip-address}
City	{city}
Country	{country}
ログイン時間	{login-time}
デバイス名	{device-name}
オンクリックリンクが有効	{one-click-link-valid}
オンクリックリンクが無効	{one-click-link-invalid}
Event ID	{event-id}
フィードバックトークン	{feedback-token}

E メールと SMS の MFA メッセージのカスタマイズ

[多要素認証 \(MFA\)](#) に関する SMS と E メールメッセージのカスタマイズするには、Amazon Cognito ユーザープールコンソールの [メッセージテンプレート] メニューで、[MFA メッセージ] を編集します。

Important

カスタムメッセージには、{####} プレースホルダーを含む必要があります。このプレースホルダーは、メッセージが送信される前に認証コードへ置き換えられます。

Amazon Cognito では、認証コードも含めて、SMS メッセージの最大長を 140 UTF-8 文字に設定しています。

SMS 検証メッセージのカスタマイズ

電話番号検証用の SMS メッセージのカスタマイズするには、ユーザープールの [メッセージテンプレート] メニューで [検証メッセージ] テンプレートを編集します。

Important

カスタムメッセージには、{####} プレースホルダーを含む必要があります。このプレースホルダーは、メッセージが送信される前に検証コードへ置き換えられます。

メッセージの最大長は検証コードを含めて UTF-8 で 140 文字です。

E メール検証メッセージのカスタマイズ

Amazon Cognito でユーザープール内のユーザーの E メールアドレスを確認するには、ユーザーが選択できるリンクを記載した E メールメッセージをユーザーに送信するか、ユーザーが入力できるコードを送信します。

E メールアドレス検証メッセージの Eメールの件名と本文のカスタマイズするには、ユーザープールの [メッセージテンプレート] メニューで [検証メッセージ] テンプレートを編集します。[検証メッセージ] テンプレートを編集するときに、[検証タイプ] として [コード] または [リンク] を選択できます。

検証タイプとしてコードを選択した場合、カスタムメッセージには {####} プレースホルダーを含める必要があります。メッセージを送信するときに、検証コードはこのプレースホルダーを置き換えます。

検証タイプとしてリンクを選択した場合、カスタムメッセージにはプレースホルダーを {##Verify Your Email##} の形式で含める必要があります。プレースホルダー文字間のテキスト文字列は、{##Click here##} のように変更できます。E メールを検証するというタイトルの検証リンクは、このプレースホルダーを置き換えます。

E メール検証メッセージのリンクは、ユーザーを次の例のような URL に誘導します。

```
https://<your user pool domain>/confirmUser/?  
client_id=abcdefg12345678&user_name=emailtest&confirmation_code=123456
```

メッセージの最大長は検証コード (ある場合) を含めて UTF-8 で 20,000 文字です。このメッセージでは、HTML タグを使用してコンテンツの書式を設定できます。

ユーザー招待メッセージのカスタマイズ

Amazon Cognito が SMS または E メールメッセージを使用して新規ユーザーに送信するユーザー招待メッセージをカスタマイズするには、[メッセージテンプレート] メニューで [招待メッセージ] テンプレートを編集できます。

Important

カスタムメッセージには、{username} および {####} のプレースホルダーを含む必要があります。Amazon Cognito が招待メッセージを送信すると、これらのプレースホルダーはユーザーのユーザー名とパスワードに置き換えられます。

SMS メッセージの最大長は検証コードを含めて UTF-8 で 140 文字です。E メールメッセージの最大長は検証コードを含めて UTF-8 で 20,000 文字です。E メールメッセージに HTML タグを使用して、コンテンツの書式を設定できます。

E メールアドレスのカスタマイズ

デフォルトでは、Amazon Cognito は、no-reply@verificationemail.com からユーザープール内のユーザーに E メールメッセージを送信します。no-reply@verificationemail.com の代わりにカスタムの FROM と REPLY-TO メールアドレスを指定するか選択できます。

FROM と REPLY-TO の E メールアドレスをカスタマイズするには

1. [\[Amazon Cognito console\]](#) (Amazon Cognito コンソール) に移動し、[User Pools] (ユーザープール) を選択します。
2. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [認証方法] メニューを選択します。[Email] (E メール) で、[Edit] (編集) を選択します。
4. [SES Region] (SES リージョン)を選択します。
5. 選択した [SES Region] (SES リージョン) の Amazon SES で検証した E メールアドレスリストから [FROM email address] (FROM E メールアドレス) を選択します。検証済みドメインの E メールアドレスを使用するには、AWS Command Line Interface または AWS API で E メール設定を設定します。詳細については、Amazon Simple Email Service デベロッパーガイドの「[Amazon SES での E メールアドレスとドメインの検証](#)」を参照してください。
6. 選択した[SES Region] (SES リージョン)の構成セットのリストから、[Configuration set] (構成セット) を選択します。
7. E メールメッセージ用に、わかりやすい [FROM sender name] (FROM 送信者名) を John Styles <johnstyles@example.com>の形式で入力します。
8. 返信先 E メールアドレスをカスタマイズするには、[REPLY-TO email address (返信先 E メールアドレス)] フィールドに有効な E メールアドレスを入力します。

Amazon SES E メールを代理送信するための Amazon Cognito の承認 (カスタム REPLY-TO E メールアドレスからの送信)

Amazon Cognito は、デフォルトのアドレスではなく、カスタムの FROM メールアドレスからメールを送信するように設定できます。カスタムアドレスを使用するには、Amazon SES で検証済みの ID から E メールメッセージを送信できる許可を Amazon Cognito に付与する必要があります。ほとんどの場合、送信承認ポリシーを作成することで許可が付与できます。詳細については、Amazon Simple Email Service デベロッパーガイドの「[Amazon SES での送信承認の使用](#)」を参照してください。

メールメッセージに Amazon SES の使用許可をユーザープールを設定すると、Amazon Cognito はアカウントに `AWSServiceRoleForAmazonCognitoIdpEmailService` ロールを作成して、Amazon SES へのアクセスを許可します。AWSServiceRoleForAmazonCognitoIdpEmailService サービスにリンクされたロールが使用される場合、送信承認ポリシーは必要ありません。ユーザープールのデフォルト E メール機能と、検証済みの Amazon SES ID の両方を FROM アドレスとして使用する場合のみ、送信承認ポリシーを追加する必要があります。

Amazon Cognito が作成するサービスリンクロールの詳細については、「[Amazon Cognito のサービスリンクロールの使用](#)」を参照してください。

次の送信承認ポリシーの例では、Amazon SES 検証済み ID を使用するための限定された権限が Amazon Cognito に付与されています。Amazon Cognito は、条件 `aws:SourceArn` のユーザープールと条件 `aws:SourceAccount` のアカウントの両方に代わって送信する場合にのみ、E メールメッセージを送信できます。その他の例については、「Amazon Simple Email Service デベロッパーガイド」の「[Amazon SES 送信承認ポリシーの例](#)」を参照してください。

Note

この例では、「Sid」値はステートメントを一意に識別する任意の文字列です。ポリシー構文の詳細については、「Amazon Simple Email Service デベロッパーガイド」の「[Amazon SES 送信承認ポリシー](#)」を参照してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "stmt1234567891234",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "email.cognito-idp.amazonaws.com"
        ]
      },
      "Action": [
        "SES:SendEmail",
        "SES:SendRawEmail"
      ],
      "Resource": "arn:aws:ses:us-east-1:111122223333:identity/support@example.com",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      },
      "ArnLike": {
```

```
        "aws:SourceArn": "arn:aws:cognito-idp:us-east-1:111122223333:userpool/us-east-1_EXAMPLE"
      }
    }
  ]
}
```

Amazon Cognito コンソールは、ドロップダウンメニューから Amazon SES の ID を選択するときに同様のポリシーを追加します。CLI または API を使用してユーザープールを設定する場合は、前の例のように構造化されたポリシーを Amazon SES アイデンティティにアタッチする必要があります。

管理者としてのユーザーアカウントの作成

ユーザープールは、インターネット上の誰でもアプリケーション内のユーザープロフィールにサインアップできるという、Customer Identity and Access Management (CIAM) のユーザーディレクトリであるだけではありません。セルフサービスサインアップを無効にすることができます。顧客の本人確認が済んでいる場合に、事前に認可された顧客のみを認めることをお勧めします。[プライベート SAML 2.0 または OIDC ID プロバイダー](#)を使用して、アプリケーションの周囲に手動認証ガードレールを配置するには、[ユーザーをインポート](#)するか、[サインアップ時にユーザーをスクリーニング](#)するか、または、管理 API オペレーションを使用してユーザーを作成します。ユーザーの管理作成ワークフローは、プログラムによることでも、別のシステムに登録した後にユーザーをプロビジョニングすることでも、Amazon Cognito コンソールでケースバイケースやテストベースで作成することでも可能です。

ユーザーを管理者として作成すると、Amazon Cognito は、ユーザーの一時的なパスワードを設定し、ウェルカムメッセージまたは招待メッセージを送信します。招待メッセージのリンクに従って初回のサインインを行い、パスワードを設定し、アカウントを確認できます。次のページでは、新しいユーザーを作成し、ウェルカムメッセージを設定する方法について説明します。ユーザープール API と AWS SDK または CDK を使用したユーザー作成の詳細については、[AdminCreateUser](#)」を参照してください。

ユーザープールを作成したら、AWS マネジメントコンソール、AWS Command Line Interface または Amazon Cognito API を使用してユーザーを作成できます。ユーザープールで新しいユーザーのプロフィールを作成し、サインアップ手順を含めたウェルカムメッセージをそのユーザーに SMS または E メールで送信できます。

管理者がユーザープール内のユーザーを管理する方法の例を以下に示します。

- Amazon Cognito コンソールまたは AdminCreateUser API オペレーションを使用して新しいユーザープロフィールを作成します。
- ユーザー名パスワード、パスワードなし、パスキー、カスタム[認証フロー](#)を、ユーザープールとアプリケーションクライアントで利用できるようにします。
- ユーザー属性値を設定します。
- カスタム属性を作成します。
- AdminCreateUser API リクエストに、イミュータブルな[カスタム属性](#)の値を設定します。この機能は Amazon Cognito コンソールでは利用できません。
- 仮パスワードを指定するか、パスワードなしでユーザーを作成するか、または Amazon Cognito にパスワードの自動生成を許可します。
- 新しいユーザーを作成し、アカウントの確認、E メールアドレスの検証、または電話番号の検証を自動的に行います。
- またはカスタムメッセージ、カスタム SMS 送信者、カスタム E メール送信者などの AWS マネジメントコンソール Lambda トリガーを使用して、新規ユーザーのカスタム SMS および [E メール招待メッセージ](#)を指定します。 [???](#)
- 招待メッセージが SMS、E メール、その両方のいずれで送信されるかを指定する。
- AdminCreateUser パラメータに RESEND を指定して、MessageAction API を呼び出すことで、既存のユーザーにウェルカムメッセージを再送信する。
- ユーザー作成時の招待メッセージの送信を[抑制](#)します。
- 新規ユーザーアカウントの有効期限を最大 90 日に指定します。
- ユーザーに自己サインアップを許可したり、管理者にのみ新しいユーザーの追加を許可したりする。

管理者は、サーバー側のアプリケーションで AWS 認証情報を使用してユーザーにサインインすることもできます。詳細については、「[API 認証と SDK 認証の認可モデル](#)」を参照してください。

ユーザー認証フローとユーザーの作成

管理者がユーザーを作成する場合、ユーザープールの設定に応じて異なるオプションを使用できます。認証フロー (ユーザーがサインインと MFA に使用できる方法) に応じて、ユーザーの作成方法とユーザーへの送信メッセージが変わる場合があります。ユーザープールで使用する認証フローをいくつか以下に示します。

- ユーザー名とパスワード
- パスキー

- サードパーティの IdP によるサインイン
- E メールおよび SMS ワンタイムパスワード (OTP) によるパスワードなしのサインイン
- E メール、SMS、Authenticator アプリケーション OTP による多要素認証
- Lambda トリガーによるカスタム認証

これらのサインイン要素の設定方法の詳細については、「[Amazon Cognito ユーザープールによる認証](#)」を参照してください。

パスワードなしでユーザーを作成する

ユーザープールでパスワードなしのサインインを有効にしている場合は、パスワードなしでユーザーを作成できます。パスワードなしでユーザーを作成するには、使用可能なパスワードなしのサインイン要素の属性値を指定する必要があります。例えば、E メール OTP によるパスワードなしのサインインがユーザープールで使用可能な場合は、パスワードなしで、E メールアドレス属性を持つユーザーを作成できます。新規ユーザーが使用できる認証フローにパスワード (パスキー、ユーザー名パスワードなど) のみが必要である場合は、新規ユーザーごとに仮パスワードを作成または生成する必要があります。

パスワードなしの新規ユーザーを作成するには

- Amazon Cognito コンソールで [パスワードを設定しない] を選択します。
- AdminCreateUser API リクエストの TemporaryPassword パラメータを省略するか、空白のままにします。

パスワードなしのユーザーは自動的に確認される

通常、新規ユーザーは作成時に仮パスワードを取得し、FORCE_CHANGE_PASSWORD ステータスになります。パスワードなしのユーザーを作成すると、すぐに CONFIRMED 状態になります。CONFIRMED 状態のユーザーに確認コードを再送信することはできません。

パスワードなしのユーザーの場合、招待メッセージは変更される

デフォルトでは、Amazon Cognito から新規ユーザーに送信される[招待メッセージ](#)は、「Your username is {userName} and your password is {####}.」という内容です。パスワードなしのユーザーを作成すると、メッセージの内容は「Your username is {userName}.」となります。ユーザーのパスワードを設定するかどうかを反映するには、招待メッセージをカスタマイズします。パスワードなしの認証モデルで {####} パスワード変数を省略します。

パスワードなしの要素が利用可能な場合、パスワードは自動生成できない

E メールまたは電話の OTP によるパスワードなしのサインインをサポートするようにユーザープールを設定している場合、パスワードを自動生成することはできません。パスワードを持つユーザーごとに、プロファイルの作成時に、仮パスワードを設定する必要があります。

パスワードなしのユーザーは、すべての必須属性の値を持っている必要がある

パスワードなしのユーザーを作成する場合、リクエストが成功するのは、ユーザープールで必須としてマークされたすべての属性の値をユーザーが提供した場合のみです。これは、OTP 配信に必要な電話番号と E メールアドレスの属性だけでなく、すべての必須属性に適用されます。

後で必須属性の値を提供するユーザーを作成する

ユーザープールで属性を必須に設定できますが、管理者によるユーザーの作成後に、アプリケーション内でユーザーとやり取りしながら、これらの属性を収集することもできます。管理者は、仮パスワードを使用してユーザーを作成するときに、必須属性の値を省略できます。パスワードなしのユーザーの場合、必須属性の値を省略することはできません。

ユーザーが必須属性の値が不足したまま仮パスワードを使用すると、初回サインイン時に [NEW_PASSWORD_REQUIRED](#) チャレンジを受け取ります。この場合、不足している必須属性の値を `requiredAttributes` パラメータで指定できます。すべての必須属性が [変更可能](#) である場合にのみ、パスワードを持つが必須属性を持たないユーザーを作成できます。ユーザーは、サインインに使用したアプリケーションクライアントから必須属性が [書き込み可能](#) である場合に限り、`NEW_PASSWORD_REQUIRED` チャレンジと必須属性の値を使用してサインインを完了できます。

管理者が作成したユーザーに永続的なパスワードを設定すると、ステータスは `CONFIRMED` に変わり、ユーザープールは初回サインイン時に新しいパスワードまたは必須属性の入力を求めません。

で新しいユーザーを作成する AWS マネジメントコンソール

Amazon Cognito コンソールを使用して、ユーザーパスワードの要件を設定し、ユーザーに送信される招待メッセージと確認メッセージを設定し、新しいユーザーを追加できます。

パスワードポリシーを設定し、自己登録を有効にする

パスワードの複雑さを最小限に抑え、ユーザープールでパブリック API を使用してユーザーがサインアップできるかどうかを設定できます。

パスワードポリシーの設定

1. [\[Amazon Cognito console\]](#) (Amazon Cognito コンソール) に移動し、[User Pools] (ユーザープール) を選択します。
2. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [認証方法] メニューを選択し、[パスワードポリシー] を見つけます。[編集] を選択します。
4. [Custom] (カスタム) の [Password policy mode] (パスワードポリシーモード) を選択します。
5. [Password minimum length] (パスワードの最小長) を選択します。パスワード長要件の制限については、「[ユーザープールのリソースクォータ](#)」を参照してください。
6. [Password complexity] (パスワードの複雑さ) 要件を選択してください。
7. 管理者が設定したパスワードの有効期間を選択します。
8. [Save changes] (変更の保存) をクリックします。

セルフサービスサインアップを許可する

1. [\[Amazon Cognito console\]](#) (Amazon Cognito コンソール) に移動し、[User Pools] (ユーザープール) を選択します。
2. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [サインアップ] メニューを選択し、[セルフサービスのサインアップ] を見つけます。[Edit] (編集) を選択します。
4. [Enable self-registration] (自己登録を有効化) するかどうかを選択します。自己登録は通常、クライアントシークレットまたは AWS Identity and Access Management (IAM) API 認証情報を配布せずにユーザープールに新しいユーザーを登録する必要があるパブリックアプリケーションクライアントで使用されます。

自己登録の無効化

自己登録を有効にしない場合は、IAM API 認証情報、またはフェデレーションプロバイダーとのサインインを使用して管理 API アクションで新しいユーザーを作成する必要があります。

5. [Save changes] (変更の保存) をクリックします。

E メールメッセージと SMS メッセージのカスタマイズ

ユーザーメッセージのカスタマイズ

Amazon Cognito がユーザーにサインインするように招待したとき、ユーザーがユーザーアカウントへのサインアップ、またはサインインして多要素認証 (MFA) を要求される際に Amazon Cognito はユーザーに送信するメッセージをカスタマイズできます。

Note

[Invitation message] (招待メッセージ) はユーザープールにユーザーを作成し、サインインに招待すると送信されます。Amazon Cognito はユーザーの E メールアドレスまたは電話番号に初期サインイン情報を送信します。

検証メッセージは、ユーザーがユーザープールのユーザーアカウントに登録したときに送信されます。Amazon Cognito はユーザーにコードを送信します。ユーザーが Amazon Cognito にコードを提供すると、連絡先情報を確認し、サインインのためにアカウントを確認します。検証コードは 24 時間有効です。

[MFA message] (MFA メッセージ) は、ユーザープールで SMS MFA を有効にし、SMS MFA を構成したユーザーがサインインして MFA を要求されたときに送信されます。

1. [\[Amazon Cognito console\]](#) (Amazon Cognito コンソール) に移動し、[User Pools] (ユーザープール) を選択します。
2. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [メッセージテンプレート] メニューを選択して、[検証メッセージ]、[招待メッセージ]、または [MFA メッセージ] を選択し、[編集] を選択します。
4. 選択したメッセージタイプのメッセージをカスタマイズします。

Note

メッセージをカスタマイズするときは、メッセージテンプレート内のすべての変数を含める必要があります。例えば、変数の場合 {#####} は含まれません。ユーザーにはメッセージアクションを完了するのに十分な情報がありません。

詳細については、「[メッセージテンプレート](#)」を参照してください。

5. a. 検証メッセージ

- i. [Email] (E メール) メッセージの [Verification type] (検証タイプ) を選択します。[Code] (コード) 検証は、ユーザーが入力する必要がある数値コードを送信します。[Link] (リンク) 検証は、ユーザーがクリックして連絡先情報を検証できるリンクを送信します。変数内のテキスト [Link] (リンク) メッセージはハイパーリンクテキストとして表示されます。例えば、変数 {# #Click here##} を使用したメッセージテンプレートは、E メールメッセージで「[Click here](#) (ここをクリックしてください)」というように表示されます。
 - ii. [Email] (E メール) メッセージの [Email subject] (Eメールの件名) を入力します。
 - iii. [Email] (E メール) メッセージのカスタムの [Email message] (Eメールメッセージ) のテンプレートを入力します。このテンプレートは HTML でカスタマイズできます。
 - iv. [SMS] メッセージのカスタムの [SMS message] (SMS メッセージ) のテンプレートを入力します。
 - v. [Save changes] (変更の保存) をクリックします。
- b. [Invitation messages] (招待メッセージ)
- i. [Email] (E メール) メッセージの [Email subject] (Eメールの件名) を入力します。
 - ii. [Email] (E メール) メッセージのカスタムの [Email message] (Eメールメッセージ) のテンプレートを入力します。このテンプレートは HTML でカスタマイズできます。
 - iii. [SMS] メッセージのカスタムの [SMS message] (SMS メッセージ) のテンプレートを入力します。
 - iv. [Save changes] (変更の保存) をクリックします。
- c. MFA メッセージ
- i. [SMS] メッセージのカスタムの [SMS message] (SMS メッセージ) のテンプレートを入力します。
 - ii. [Save changes] (変更の保存) をクリックします。

ユーザーの作成

ユーザーの作成

Amazon Cognito コンソールからユーザープールの新しいユーザーを作成できます。通常、ユーザーはパスワードを設定した後にサインインできます。E メールアドレスでサインインするには、ユーザーは email 属性を確認する必要があります。電話番号でサインインするには、ユーザーは phone_number 属性を確認する必要があります。アカウントを管理者として確認するには、AWS

CLI または API を使用するか、フェデレーティッド ID プロバイダーでユーザープロフィールを作成することもできます。詳細については、「[Amazon Cognito API Reference](#)」を参照してください。

1. [\[Amazon Cognito console\]](#) (Amazon Cognito コンソール) に移動し、[User Pools] (ユーザープール) を選択します。
2. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
3. [ユーザー] メニュー、[ユーザーを作成] の順に選択します。
4. パスワード要件、使用可能なアカウント回復方法、およびユーザープールのエイリアス属性に関するガイダンスは、「ユーザープールのサインインとセキュリティ要件」を参照してください。
5. 招待メッセージの送信方法を選択します。SMS メッセージ、E メールメッセージ、または両方を選択します。招待メッセージを抑制するには、[招待を送信しない] を選択します。

 Note

招待メッセージを送信する前に、ユーザープールの [認証方法] メニューで、Amazon Simple Notification Service および Amazon Simple Email Service により、送信者と AWS リージョン を設定します。受信者メッセージとデータレートが適用されます。Amazon SES は、メールメッセージの請求を別途請求し、Amazon SNS は SMS メッセージについて別途請求します。

6. 新規ユーザー用に [Username] (ユーザー名) を選択します。
7. [Create a password] (パスワードを作成する) または、Amazon Cognito にユーザーの [Generate a password] (パスワードを生成する) を許可するかどうかを選択してください。[\[パスワードなしのサインイン\]](#) がユーザープールで利用可能な場合、パスワードを生成するオプションは使用できません。一時パスワードは、ユーザープールのパスワードポリシーに準拠する必要があります。
8. [作成] を選択します。
9. [ユーザー] メニューを選択し、[ユーザー名] でユーザー名を入力します。[User attributes] (ユーザー属性) および [Group memberships] (グループメンバーシップ) を追加して編集します。ユーザーイベント履歴の確認

ユーザープールにグループを追加する

Amazon Cognito ユーザープール内のグループのサポートにより、ユーザーの作成と管理、グループへのユーザーの追加、およびグループからのユーザーの削除が可能になります。グループを使用して、ユーザーのコレクションを作成してそのアクセス権限を管理したり、異なるタイプのユーザー

を表したりできます。AWS Identity and Access Management (IAM) ロールをグループに割り当てると、グループのメンバーのアクセス許可を定義できます。

グループを使用して、ユーザープールでユーザーのコレクションを作成できます。この操作は、これらのユーザーのアクセス権限を設定するためによく行われます。例えば、ウェブサイトやアプリの読者、寄稿者、および編集者であるユーザーに対して別個のグループを作成できます。また、グループに関連付けられた IAM ロールを使用することで、これらの異なるグループに異なる許可を設定して、コントリビュータのみがコンテンツを Amazon S3 に配置し、エディタのみが Amazon API Gateway の API を通じてコンテンツをパブリッシュできるようにすることも可能です。

Amazon Cognito は、ユーザープールに追加する OIDC、SAML、ソーシャル [ID プロバイダー](#) (IdP) ごとにユーザーグループを作成します。グループ名の形式は [user pool ID]_[IdP name] です (例: us-east-1_EXAMPLE_MYSSO、us-east-1_EXAMPLE_Google)。自動生成された一意の各 IdP ユーザープロフィールは、このグループに自動的に追加されます。[リンクされたユーザー](#)は、このグループに自動的に追加されませんが、そのプロフィールを別のプロセスでグループに追加できます。

ユーザープールでグループを作成および管理するには AWS マネジメントコンソール、APIs、および CLI を使用します。開発者 (AWS 認証情報を使用) は、ユーザープールのグループを作成、読み取り、更新、削除、一覧表示できます。また、グループに対してユーザーを追加、削除できます。

ユーザープール内のグループを使用しても追加コストは発生しません。詳細については、「[Amazon Cognito の料金](#)」を参照してください。

グループへの IAM ロールの割り当て

グループを使用して、IAM ロールでリソースへの許可を制御できます。IAM ロールには、信頼ポリシーと許可ポリシーが含まれます。ロールの [信頼](#) ポリシーでは、ロールを使用できるユーザーを指定します。[アクセス許可](#) ポリシーでは、グループメンバーがアクセスできるアクションとリソースを指定します。IAM ロールを作成するときに、グループユーザーがロールを引き受けることを許可するロールの信頼ポリシーを設定します。ロールの許可ポリシーで、グループに付与する許可を指定します。

Amazon Cognito でグループを作成するときは、ロールの [ARN](#) を指定して IAM ロールを指定します。グループメンバーが Amazon Cognito を使用してサインインすると、ID プールから一時的な認証情報を受け取ることができます。それらの許可は、関連付けられた IAM ロールによって決まります。

各ユーザーは複数のグループに属することができます。デベロッパーとして、ユーザーが複数のグループに属している場合に IAM ロールを自動的に選択するための以下のオプションがあります。

- 各グループに優先順位の値を割り当てることができます。優先順位が高い (低い) グループが選択され、それに関連付けられた IAM ロールが適用されます。
- また、アプリは、ID プールを通じてユーザーの AWS 認証情報をリクエストするとき、[GetCredentialsForIdentity](#) CustomRoleARN パラメータでロール ARN を指定して、利用できるロールから選択できます。指定された IAM ロールは、ユーザーが利用できるロールに一致する必要があります。

グループへの優先順位の値の割り当て

ユーザーは複数のグループに属することができます。ユーザーのアクセストークンと ID トークンの `cognito:groups` クレームには、ユーザーが属するすべてのグループのリストが含まれます。`cognito:roles` クレームには、グループに対応するロールのリストが含まれます。

ユーザーは複数のグループに属することができるので、各グループに優先順位を割り当てることができます。これは、ユーザーがユーザープールで属するその他のグループに対するこのグループの優先順位を指定する正数です。ゼロが最優先順位値です。低い優先順位の値を持つグループは、高い優先順位の値または null 値を持つグループよりも優先されます。ユーザーが複数のグループに属している場合、ユーザーの ID トークンの `cognito:preferred_role` クレームに適用される IAM ロールは、優先順位の値が最も低いグループのものになります。

2 つのグループは、同じ優先順位の値を持つことができます。その場合は、どちらのグループも他方に対して優先されません。同じ優先順位の値を持つ 2 つのグループのロール ARN が同じである場合、そのロールは、各グループのユーザーの ID トークンの `cognito:preferred_role` クレームで使用されます。2 つのグループのロール ARN が異なる場合、`cognito:preferred_role` クレームは、ユーザーの ID トークンで設定されません。

Amazon API Gateway を使用した許可の管理でのグループの使用

Amazon API Gateway を使用した許可の管理には、ユーザープールのグループを使用することができます。ユーザーがメンバーであるグループは、ID トークンと `cognito:groups` クレームのユーザープールからのアクセストークンの両方に含まれています。リクエストとともに ID またはアクセストークンを Amazon API Gateway に送信し、REST API に Amazon Cognito ユーザープールオーソライザーを使用できます。詳細については、[API Gateway デベロッパーガイド](#)の「[Amazon Cognito ユーザープールをオーソライザーとして使用して REST API へのアクセスを制御する](#)」を参照してください

カスタム JWT オーソライザーを使用して Amazon API Gateway HTTP API へのアクセスを承認することもできます。詳細については、[API Gateway デベロッパーガイド](#)の「[JWT オーソライザーを使用した HTTP API へのアクセスの制御](#)」を参照してください。

グループの制限

ユーザーグループには、次の制限が適用されます。

- 作成できるグループの数は、[Amazon Cognito サービスクォータ](#)によって制限されます。
- グループはネストできません。
- グループのユーザーを検索することはできません。
- グループを名前で検索することはできませんが、グループをリストすることはできます。

で新しいグループを作成する AWS マネジメントコンソール

以下の手順に従って、新しいグループを作成します。

新しいグループを作成する

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [グループ] メニュー、[グループを作成] の順に選択します。
5. [Create a group] (グループを作成する) ページで、[Group name] (グループ名) に新しいグループのフレンドリ名を入力します。
6. オプションで、次のフィールドのいずれかを使用して、このグループに関する追加情報を指定できます。
 - [Description] (説明) - この新しいグループの使用目的の詳細を入力します。
 - [Precedence] (優先順位) - Amazon Cognito は、優先度が低いグループに属するグループに基づいて、特定のユーザーに対するすべてのグループ権限を評価し、適用します。優先順位が低いグループが選択され、それに関連付けられた IAM ロールが適用されます。詳細については、「[グループへの優先順位の値の割り当て](#)」を参照してください。
 - [IAM role] (IAM ロール) - リソースへのアクセス許可を制御する必要がある場合は、グループに IAM ロールを割り当てることができます。ID プールを持つユーザープールを統合する場合、

[IAM ロール] の設定によって、トークンからロールを選択するよう ID プールが設定されているときに、ユーザーの ID トークンに割り当てられるロールが決定されます。詳細については、「[グループへの IAM ロールの割り当て](#)」を参照してください。

- [Add users to this group] (ユーザーをこのグループに追加する) - 既存のユーザーを作成した後、このグループのメンバーとして追加します。

7. [Create] (作成) を選択して確定します。

ユーザーアカウントの管理と検索

ユーザープールには数百万のユーザーを含めることができます。このサイズのデータセットを使用するのは管理者にとって課題になります。Amazon Cognito には、ユーザープロフィールの検索や変更を行うツールがあります。ユーザーを検索する主な方法は、Amazon Cognito コンソールの [ユーザー] メニューと [ListUsers](#) を使用することです。ユーザーに関する情報を取得する方法としては、[AdminGetUser](#) などとは異なり、コストに影響しないオプションです。

本ガイドのこのセクションには、ユーザープール内のユーザープロフィールの検索と更新に関する情報が記載されています。

ユーザー属性の表示

Amazon Cognito コンソールでユーザー属性を表示するには、以下の手順に従います。

ユーザー属性を表示するには

1. [Amazon Cognito コンソール](#) に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択し、リストからユーザーを選択します。
5. ユーザーの詳細ページの [User attributes] (ユーザー属性) で、ユーザーに関連付けられている属性を表示できます。

ユーザーのパスワードのリセット

Amazon Cognito コンソールでユーザーのパスワードをリセットするには、以下の手順に従います。

ユーザーのパスワードのリセット

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択し、リストからユーザーを選択します。
5. ユーザーの詳細ページで、[Actions] (アクション)、[Reset password] (パスワードのリセット) を選択します。
6. [Reset password] (パスワードのリセット) ダイアログで、情報を確認し、準備ができたなら [Reset] (リセット) を選択します。

この行動により、確認コードがユーザーに即時送信され、ユーザーの状態が RESET_REQUIRED に変更されることでユーザーの現在のパスワードが無効にされます。[Reset password] (ユーザーパスワードのリセット) コードは、1 時間有効です。

ユーザーアカウントの有効化、無効化、削除

使用されていないユーザープロフィールを削除できます。一時的にアクセスを制限する場合は、無効にすることもできます。ユーザーは自分のアカウントを削除できますが、ユーザーアカウントを有効または無効にできるのはユーザープール管理者のみです。

削除の影響

ユーザーは、削除済みのユーザーアカウントでサインインすることはできません。アクセスを回復するには、サインアップするか、再度作成する必要があります。

アカウントの無効化の影響

ユーザーアカウントを無効にすると、Amazon Cognito はすべての認証済みセッションを自動的に無効にし、サインイン用のユーザーアカウントを非アクティブ化して、[アクセストークンと更新トークンの取り消し](#)を行います。無効化したアカウントにユーザーがサインインしようすると、Amazon Cognito は invalid_request エラーとともにメッセージ User is not enabled を返します。この動作は、アプリケーションクライアントの[ユーザー存在の開示設定](#)では変更されません。ローカルユーザーアカウントと、フェデレーションユーザーアカウントのローカルプロフィールを無効にすることができます。ユーザーがマネージドログインまたはクラシックのホストされた UI を使用してサインインし、その後アカウントを無効にした場合、認証されたセッションを維持するブラウザ

Cookie を通じて再度サインインしようとする、Amazon Cognito によってログインページにリダイレクトされます。

アカウントの有効化の影響

ユーザーは、アカウントを有効にすると、すぐにサインインできます。ユーザーアカウントはデフォルトで有効になります。ユーザーの属性とパスワードは、アカウントが無効になる前と同じままです。アプリケーションで取り消したトークンは、ユーザーアカウントが無効にしたか、更新トークンを別個に取り消したかにかかわらず、トークンを所有していたユーザーアカウントを有効にした後も無効のままになります。

Delete a user account (console)

ユーザーアカウントを削除するには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択し、リスト内のユーザーのユーザー名の横にあるラジオボタンを選択します。
5. [削除] を選択します。
6. [アクセスの有効化] を選択します。
7. [削除] を選択します。

Delete a user account (API)

ユーザーは、セルフサービスのアクセストークンで認可された [DeleteUser](#) API オペレーションを使用してアカウントを削除できます。DeleteUser リクエスト本文の例を次に示します。

```
{
  "AccessToken": "eyJra456defEXAMPLE"
}
```

管理者は、IAM で認可された [AdminDeleteUser](#) API オペレーションを使用してユーザーアカウントを削除できます。AdminDeleteUser リクエスト本文の例を次に示します。

```
{
```

```
{
  "Username": "testuser",
  "UserPoolId": "us-west-2_EXAMPLE"
}
```

Disable a user account (console)

ユーザーアカウントを無効にするには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択し、リストからユーザーのユーザー名を選択します。
5. ユーザーの詳細ページで、[アクション]、[ユーザーアクセスの無効化] の順に選択します。
6. 表示されたダイアログで、[無効化] を選択します。

Disable a user account (API)

管理者は、IAM で認可された [AdminDisableUser](#) API オペレーションを使用してユーザーアカウントを無効化できます。AdminDisableUser リクエスト本文の例を次に示します。

```
{
  "Username": "testuser",
  "UserPoolId": "us-west-2_EXAMPLE"
}
```

Enable a user account (console)

ユーザーアカウントを有効にするには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択し、リストからユーザーのユーザー名を選択します。
5. ユーザーの詳細ページで、[アクション]、[ユーザーアクセスの有効化] の順に選択します。
6. 表示されたダイアログで、[有効化] を選択します。

Enable a user account (API)

管理者は、IAM で認可された [AdminEnableUser](#) API オペレーションを使用してユーザーアカウントを有効化できます。AdminEnableUser リクエスト本文の例を次に示します。

```
{
  "Username": "testuser",
  "UserPoolId": "us-west-2_EXAMPLE"
}
```

ユーザー属性の検索

ユーザープールがすでに作成されている場合は、AWS マネジメントコンソールの [Users] (ユーザー) パネルから検索できます。また、[Filter] (フィルター) パラメータを受け入れる Amazon Cognitoの [ListUsers API](#) も使用できます。

以下のどの標準属性でも検索できます。カスタム属性は検索できません。

- username (大文字と小文字が区別されます)
- email
- phone_number
- name
- given_name
- family_name
- preferred_username
- cognito:user_status (コンソールでは [Status (ステータス)] となっています) (大文字と小文字が区別されません)
- status (コンソールでは [Enabled (有効)] となっています) (大文字と小文字が区別されます)
- sub

Note

また、クライアント側のフィルターを使用してユーザーを一覧表示することもできます。サーバー側のフィルターは、1 つ以上の属性に一致しません。高度な検索を行うには、AWS Command Line Interfaceでの `list-users` のアクションの `--query` パラメータを用いたクライアント側のフィルターを使用します。クライアント側のフィルターを使用する

と、ListUsers は 0 人以上のユーザーからなるページ分割されたリストを返します。結果がゼロで、複数のページを連続して受信できます。Null ページネーショントークン値を受け取るまで、返される各ページネーショントークンでクエリを繰り返し、組み合わせた結果を確認します。

サーバー側およびクライアント側のフィルタリングの詳細については、AWS Command Line Interface 「ユーザーガイド」の「[出力のフィルタリング AWS CLI](#)」を参照してください。

でユーザーを検索する AWS マネジメントコンソール

ユーザープールがすでに作成されている場合は、AWS マネジメントコンソールの [Users] (ユーザー) パネルから検索できます。

AWS マネジメントコンソール 検索は常にプレフィックス (「starts with」) 検索です。

Amazon Cognito コンソールでユーザーを検索する

1. [Amazon Cognito コンソール](#)に移動します。AWS 認証情報の入力を求められる場合があります。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択し、検索フィールドにユーザー名を入力します。[Username] (ユーザーネーム) など、一部の属性値では、大文字と小文字が区別される点に注意してください。

また、検索フィルターを調整して、スコープを他のユーザープロパティ ([Email] (E メール)、[Phone number] (電話番号)、または [Last name] (姓) など) に絞り込むことで、ユーザーを検索することもできます。

ListUsers API を使用したユーザーの検索

アプリからユーザーを検索するには、Amazon Cognito の [ListUsers API](#) を使用します。この API は以下のパラメータを使用します。

- AttributesToGet: 文字列の配列です。各文字列は、検索結果でユーザーごとに返されるユーザー属性の名前です。すべての属性を取得するには、AttributesToGet パラメータを含めないか、またはリテラル文字列の値 null を使って AttributesToGet をリクエストしてください。

- **Filter:** 「'''」形式のフィルター文字列です。AttributeNameFilter-TypeAttributeValue フィルター文字列内の引用符は、円記号 (\) を使用してエスケープする必要があります。例えば、`"family_name = \"Reddy\""`。フィルター文字列が空の場合、ListUsers はユーザープールのすべてのユーザーを返します。
- **AttributeName:** 検索する属性の名前。同時に検索できる属性は 1 つのみです。

 Note

標準属性のみを検索できます。カスタム属性は検索できません。これはインデックスが付けられた属性のみが検索可能なため、カスタム属性にインデックスを作成することはできません。

- **Filter-Type:** 完全一致を検索する場合は、`=` を使用します (`given_name = "Jon"` など)。プレフィックス (「先頭の文字」) 一致を検索する場合、`^=` を使用します (`given_name ^= "Jon"` など)。
- **AttributeValue:** ユーザーごとに一致する必要がある属性値です。
- **Limit:** 返されるユーザーの最大数です。
- **PaginationToken:** 前の検索からさらに結果を取得するためのトークンです。Amazon Cognito は 1 時間後にページ分割トークンを期限切れにします。
- **UserPoolId:** 検索を実行する必要があるユーザープールのユーザープール ID です。

すべての検索で大文字と小文字が区別されません。検索結果は、AttributeName 文字列により指定された属性によって昇順に並べ替えられます。

ListUsers API の使用例

次の例はすべてのユーザーを返します。すべての属性が含まれています。

```
{
  "AttributesToGet": null,
  "Filter": "",
  "Limit": 10,
  "UserPoolId": "us-east-1_samplepool"
}
```

次の例は、電話番号の先頭が「+1312」のすべてのユーザーを返します。また、すべての属性が含まれています。

```
{
  "AttributesToGet": null,
  "Filter": "phone_number ^= \"+1312\"",
  "Limit": 10,
  "UserPoolId": "us-east-1_samplepool"
}
```

次の例は、姓が「Reddy」の先頭 10 人のユーザーを返します。各ユーザーの検索結果には、ユーザーの名、電話番号、E メールアドレスが含まれています。ユーザープールに存在する一致ユーザーが 10 人を超える場合、レスポンスにはページ分割トークンが含まれています。

```
{
  "AttributesToGet": [
    "given_name",
    "phone_number",
    "email"
  ],
  "Filter": "family_name = \"Reddy\"",
  "Limit": 10,
  "UserPoolId": "us-east-1_samplepool"
}
```

前の例でページ分割トークンが返された場合、次の例は同じフィルター文字列と一致する次の 10 人のユーザーを返します。

```
{
  "AttributesToGet": [
    "given_name",
    "phone_number",
    "email"
  ],
  "Filter": "family_name = \"Reddy\"",
  "Limit": 10,
  "PaginationToken": "pagination_token_from_previous_search",
  "UserPoolId": "us-east-1_samplepool"
}
```


パスワード、アカウント復旧、パスワードポリシー

ユーザープールにサインインするすべてのユーザーには、それが[フェデレーションユーザー](#)であっても、ユーザープロフィールにパスワードが割り当てられます。[ローカルユーザー](#)と[リンクされたユーザー](#)は、サインイン時にパスワードを指定する必要があります。フェデレーションユーザーは、ユーザープールのパスワードを使用しないで、ID プロバイダー (IdP) でサインインします。ユーザー独自のパスワードのリセット、管理者としてのパスワードのリセットや変更、パスワードの複雑さや履歴に関する[ポリシーの設定](#)を、ユーザーに許可できます。

Amazon Cognito はユーザーパスワードをプレーンテキストで保存しません。代わりに、各ユーザーのパスワードのハッシュをユーザー固有の Salt とともに保存します。このため、ユーザープールのユーザープロフィールから既存のパスワードを取得できません。ベストプラクティスとして、プレーンテキストのユーザーパスワードはどこにも保存しないでください。ユーザーがパスワードを忘れたときにパスワードリセットを実行します。

パスワードのリセットと復旧

ユーザーがパスワードを忘れることがあります。その場合、パスワードのリセットをユーザー自らができるようにしておくことや、管理者がパスワードのリセットをユーザーに要求することが考えられます。Amazon Cognito ユーザープールでは、両方のモデルのオプションを用意しています。本ガイドのこのパートでは、ユーザープールの設定と、パスワードリセットの API オペレーションについて説明します。

[ForgotPassword](#) API オペレーションと、マネージドログインの [パスワードを忘れた場合] オプションでは、ユーザーが正しいコードを持っていることを確認したときに、[ConfirmForgotPassword](#) を使用して、新しいパスワードを設定できるコードを送信します。これはセルフサービスのパスワード復旧モデルです。

未検証ユーザーの復旧

E メールアドレスまたは電話番号を確認したユーザーには、復旧メッセージを送信できます。ユーザーが確認済みの復旧 E メールまたは電話を持っていない場合、ユーザープール管理者はユーザーの E メールアドレスまたは電話番号を検証済みとしてマークできます。Amazon Cognito コンソールでユーザーのユーザー属性を編集し、[電話番号を検証済みとしてマークする] または [E メールアドレスを検証済みとしてマークする] の横にあるチェックボックスをオンにします。または、[AdminUpdateUserAttributes](#) リクエストで email_verified または phone_number_verified を true に設定することもできます。新規ユーザーの場合は、[ResendConfirmationCode](#) API オペレーションからユーザーの E メールアドレスまたは電話番号に新しいコードが送信され、ユーザーはセルフサービスで確認と検証を完了できます。

管理者としてパスワードをリセットする

[AdminSetUserPassword](#) API オペレーションと [AdminResetUserPassword](#) API オペレーションは、管理者が開始するパスワードリセット方法です。AdminSetUserPassword は、一時的または永続的なパスワードを設定し、AdminResetUserPassword は、ForgotPassword と同じ方法でパスワードリセットコードをユーザーに送信します。

パスワードのリセットと復旧を設定する

Amazon Cognito は、コンソールでのユーザープールの作成時に選択した必須属性とサインインオプションに基づいて、アカウント復旧オプションを自動的に選択します。デフォルト設定は変更できません。

ユーザーが希望する MFA の方法は、パスワードの復旧に使用できる方法に影響します。希望する MFA を E メールメッセージにしたユーザーは、パスワードリセットコードを E メールで受信できません。希望する MFA を SMS メッセージにしたユーザーは、パスワードリセットコードを SMS で受信できません。

希望するパスワードリセット方法の対象にユーザーがいない場合、[\[パスワード復旧\]](#) 設定で代替オプションを提供する必要があります。例えば、復旧メカニズムで E メールが最優先事項になっており、E メール MFA がユーザープールのオプションである場合があります。この場合、SMS メッセージによるアカウント復旧を 2 番目のオプションとして追加するか、管理 API オペレーションを使用してこれらのユーザーのパスワードをリセットします。

Amazon Cognito は、有効な復旧方法を持たないユーザーからのパスワードリセットリクエストに対して、InvalidParameterException エラーレスポンスで応答します。

Note

ユーザーは、MFA とパスワードのリセットコードを、同じ E メールアドレスや電話番号で受け取ることはできません。E メールメッセージのワンタイムパスワード (OTP) を MFA に使用する場合、アカウントの復旧には SMS メッセージを使用する必要があります。SMS メッセージの OTP を MFA に使用する場合、アカウントの復旧には E メールメッセージを使用する必要があります。MFA を使用するユーザープールでは、属性として E メールアドレスがあっても電話番号がないか、電話番号があっても E メールアドレスがない場合、ユーザーはセルフサービスのパスワード復旧を完了できない可能性があります。

この設定でユーザーがユーザープールのパスワードをリセットできない状況を防ぐには、email および phone_number [属性を必須](#)に設定します。別の方法として、ユーザーのサインアップ時や管理者によるユーザープロフィールの作成時に、これらの属性を常に収集して設定するようにプロセスを設定することもできます。ユーザーが両方の属性を持ってい

る場合、Amazon Cognito は、ユーザーの MFA 要素ではない送信先にパスワードリセットコードを自動的に送信します。

次の手順では、ユーザープールでセルフサービスのアカウント復旧を設定します。

Configure self-service password reset (API/SDK)

AccountRecoverySetting パラメータは、ユーザーが [ForgotPassword](#) API リクエストを使用するか、マネージドログインで [パスワードを忘れた場合] を選択したときに、パスワードの復旧に使用できる方法を設定するユーザープールパラメータです。ForgotPassword は、検証済みの E メールまたは検証済みの電話番号に復旧コードを送信します。回復用コードは 1 時間有効です。ユーザープールに [AccountRecoverySetting](#) を指定すると、Amazon Cognito は、設定した優先度に基づいてコードの配信先を選択します。

AccountRecoverySetting を定義し、ユーザーに SMS MFA が設定されている場合、SMS をアカウント復旧メカニズムとして使用することはできません。この設定の優先度は、1 を最も高い優先度として決定されます。Amazon Cognito は、指定された方法の 1 つだけに検証を送信します。次の AccountRecoverySetting 例では、アカウント復旧コードのプライマリ送信先として E メールアドレスを設定します。ユーザーに E メールアドレス属性がない場合は SMS メッセージにフォールバックします。

```
"AccountRecoverySetting": {
  "RecoveryMechanisms": [
    {
      "Name": "verified_email",
      "Priority": 1
    },
    {
      "Name": "verified_phone_number",
      "Priority": 2
    }
  ]
}
```

この値 `admin_only` では、セルフサービスのアカウント復旧がオフになるため、ユーザーは代わりに管理者に連絡してパスワードをリセットする必要があります。他のアカウント復旧メカニズムでは `admin_only` を使用できません。次の例を参照してください。

```
"AccountRecoverySetting": {
```

```
"RecoveryMechanisms": [  
  {  
    "Name": "admin_only",  
    "Priority": 1  
  }  
]  
}
```

AccountRecoverySetting を指定しない場合、Amazon Cognito は、まず検証済みの電話番号に復旧コードを送信します。ユーザーに電話番号属性がない場合は、検証済みの E メールアドレスに送信します。

AccountRecoverySetting の詳細については「[CreateUserPool](#)」と「[UpdateUserPool](#)」を参照してください。

Configure self-service password reset (console)

ユーザープールの [サインイン] メニューから、アカウント復旧オプションとパスワードリセットオプションを設定します。

ユーザーアカウントの復旧を設定するには

1. [Amazon Cognito コンソール](#)にサインインします。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [サインイン] メニューを選択します。[ユーザーアカウントの復旧] を見つけて[編集] を選択します
5. ユーザーが自分のパスワードをリセットできるようにするには、[セルフサービスのアカウントの復旧を有効化] を選択します。
6. ユーザープールからユーザーに送信する、パスワードの復旧コードの配信方法を設定します。[ユーザーアカウントの復旧メッセージの配信方法] で、使用可能なオプションを選択します。ベストプラクティスとして、メッセージの送信にセカンダリ方法があるオプション (利用可能な場合は E メール、それ以外の場合は SMS など) を選択します。セカンダリ配信方法を使用すると、Amazon Cognito は MFA とは異なる方法でパスワードをリセットするようユーザーにコードを送信できます。
7. [変更を保存] を選択します。

パスワードを忘れた場合の対応

ユーザーがパスワードを忘れた場合や忘れたパスワードを確認する場合のアクションの一環として、パスワードのリセットコードをリクエストしたり、入力したりする試行を所定時間内に 5~20 回許可します。正確な回数は、リクエストに関連付けられたリスクパラメータによって異なります。この対応は変更される場合があることに注意してください。

ユーザープールのパスワードの追加要件

強力で複雑なパスワードは、ユーザープールのセキュリティ上のベストプラクティスです。特にインターネットに公開されているアプリケーションでは、パスワードが弱いと、パスワードを推測してデータにアクセスしようとするシステムに対して、ユーザーの認証情報が漏洩する可能性があります。パスワードが複雑であればあるほど、推測しにくくなります。Amazon Cognito には、[脅威保護](#)や [AWS WAF ウェブ ACL](#) など、セキュリティを重視する管理者向けの追加のツールがありますが、パスワードポリシーは、ユーザーディレクトリのセキュリティにおいて中心となる要素です。

Amazon Cognito ユーザープールのローカルユーザーのパスワードは、自動的に期限切れになることはありません。ベストプラクティスとして、ユーザーパスワードのリセットの日時やメタデータを外部システムでログに記録します。パスワード利用期間の外部ログを使用すると、アプリケーションまたは Lambda トリガーは、ユーザーのパスワード利用期間を検索し、特定の期間が経過した後にリセットを要求できます。

ユーザープールは、セキュリティ標準に準拠するパスワードの最小限の複雑さを要求するように設定できます。複雑なパスワードは、長さが 8 文字以上です。大文字、数字、特殊文字を組み合わせたパスワードも複雑なパスワードに含まれます。

エッセンシャルまたはプラス機能階層では、パスワードの再利用ポリシーを設定することもできます。ユーザーによるパスワードのリセットにあたって、新しいパスワードの設定を、現在のパスワードに加えて、追加で最大 23 個の以前のパスワード (合計 24 個までのパスワード) に一致できないようにすることができます。

ユーザープールのパスワードポリシーを設定するには

1. ユーザープールを作成して [セキュリティ要件を設定] ステップに移動するか、既存のユーザープールにアクセスして [認証方法] メニューに移動します。
2. [パスワードポリシー] に移動します。
3. [パスワードポリシーモード] を選択します。[Cognito のデフォルト] では、推奨最小設定でユーザープールを設定します。[カスタム] パスワードポリシーを選択することもできます。

4. [パスワードの最小文字数] を設定します。すべてのユーザーは、この値以上の長さのパスワードでサインアップまたは作成する必要があります。この最小値は 99 まで設定できますが、ユーザーは最大 256 文字のパスワードを設定できます。
5. [パスワードの要件] でパスワードの複雑さのルールを設定します。各ユーザーのパスワードで少なくとも 1 つは必須とする文字タイプ (数字、特殊文字、大文字、小文字) を選択します。

パスワードに以下の文字が最低 1 個は含まれることを要件として設定できます。Amazon Cognito がパスワードに最低限の必須文字が含まれていることを検証したら、ユーザーのパスワードには、任意のタイプで追加の文字を、パスワードの最大の長さまで含めることができます。

- 大文字と小文字の [基本的なラテン](#) 文字
- 数字
- 以下の特殊文字。

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

- 先頭でも末尾にもない空白文字。
6. [管理者が設定した一時パスワードの有効期限] に値を設定します。この期間が過ぎると、Amazon Cognito コンソールまたは AdminCreateUser で作成した新しいユーザーは、サインインして新しいパスワードを設定できなくなります。一時パスワードでサインインすると、ユーザーアカウントが期限切れになることはありません。Amazon Cognito ユーザープール API のパスワード有効期間を更新するには、[CreateUserPool](#) または [UpdateUserPool](#) API リクエストで [TemporaryPasswordValidityDays](#) の値を設定します。
 7. 可能な場合は、[以前のパスワードの使用を防止] の値を設定します。この機能を使用するには、ユーザープールでエッセンシャルまたはプラス [機能階層](#) を選択します。このパラメータの値は、ユーザーによるパスワードのリセット時に新しいパスワードの設定で一致させないようにする以前のパスワードの数です。

期限切れのユーザーアカウントのアクセス権をリセットするには、以下のいずれかを実行します。

- 新しい仮パスワードを送信し、MessageAction が RESEND に設定されている [AdminCreateUser](#) API リクエストを使用して有効期限をリセットします。
- ユーザープロフィールを削除し、新しいものを作成します。
- [AdminResetUserPassword](#) API リクエストで新しい確認コードを生成します。

ユーザープールへのユーザーのインポート

既存のユーザーディレクトリまたはユーザーデータベースから Amazon Cognito にユーザーをインポートまたは移行する方法は 2 つあります。1 つは、ユーザー移行の Lambda トリガーを使用して、ユーザーが Amazon Cognito を使用して初めてサインインするときにユーザーを移行する方法です。この方法では、ユーザーが既存のパスワードを引き続き使用でき、ユーザープールへの移行後にパスワードをリセットする必要はありません。もう 1 つは、すべてのユーザーのユーザープロフィール属性を含む CSV ファイルをアップロードして、ユーザーを一括で移行する方法です。以降のセクションでは、これらの両方のアプローチについて説明します。

その他のリソース

- [Amazon Cognito ユーザープールへのユーザー移行方法](#)
- [AWS re:Inforce 2023 - Amazon Cognito への移行](#)

トピック

- [ユーザー移行の Lambda トリガーを使用したユーザーのインポート](#)
- [CSV ファイルからユーザープールへのユーザーのインポート](#)

ユーザー移行の Lambda トリガーを使用したユーザーのインポート

このアプローチでは、ユーザーがアプリケーションで初めてサインインするとき、またはパスワードのリセットをリクエストするときに、既存のユーザーディレクトリからユーザープールにユーザーをシームレスに移行できます。[ユーザー移行の Lambda トリガー](#) 関数をユーザープールに追加すると、サインインしようとするユーザーに関するメタデータを受け取り、外部 ID ソースからユーザープロフィール情報を返します。この Lambda トリガーの詳細 (リクエストとレスポンスのパラメータなど) とサンプルコードについては、「[ユーザー移行の Lambda トリガーのパラメータ](#)」を参照してください。

ユーザーの移行を開始する前に、AWS アカウントでユーザー移行の Lambda 関数を作成し、ユーザープールに Lambda 関数をユーザー移行トリガーとして設定します。Lambda 関数の呼び出しと、独自のユーザープールのコンテキスト内でのみの実行を、Amazon Cognito サービスアカウントプリンシパルである `cognito-idp.amazonaws.com` のみに許可する認可ポリシーを Lambda 関数に追加します。詳細については、「[AWS Lambda のリソースベースのポリシーを使用する \(Lambda 関数ポリシー\)](#)」を参照してください。

サインインプロセス

1. ユーザーがアプリケーションを開き、Amazon Cognito ユーザープール API またはマネージドログインを使用してサインインします。Amazon Cognito API でサインインを容易にする方法の詳細については、「[Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#)」を参照してください。
2. アプリケーションはユーザー名とパスワードを Amazon Cognito に送信します。アプリに AWS SDK で構築したカスタムサインイン UI がある場合、アプリは `USER_PASSWORD_AUTH` または `ADMIN_USER_PASSWORD_AUTH` フローで [InitiateAuth](#) または [AdminInitiateAuth](#) を使用する必要があります。アプリがこれらのフローのいずれかを使用する場合、SDK はパスワードをサーバーに送信します。

 Note

ユーザー移行トリガーを追加する前に、アプリケーションクライアントの設定で `USER_PASSWORD_AUTH` または `ADMIN_USER_PASSWORD_AUTH` フローを有効化します。デフォルトの `USER_SRP_AUTH` フローの代わりにこれらのフローを使用する必要があります。Amazon Cognito は、他のディレクトリでのユーザーの認証を検証できるように、Lambda 関数にパスワードを送信する必要があります。SRP は、Lambda 関数からユーザーのパスワードを隠します。

3. Amazon Cognito は、送信されたユーザー名がユーザープール内のユーザー名またはエイリアスと一致するかどうかをチェックします。ユーザーの電子メールアドレス、電話番号、または優先ユーザー名を、ユーザープールのエイリアスとして設定できます。ユーザーが存在しない場合、Amazon Cognito は、ユーザー名やパスワードなどのパラメータを [ユーザー移行の Lambda トリガー](#) 関数に送信します。
4. [ユーザー移行の Lambda トリガー](#) 関数は、既存のユーザーディレクトリまたはユーザーデータベースを使用してユーザーをチェックまたは認証します。この関数は、Amazon Cognito がユーザープールのユーザープロファイルに保存するユーザー属性を返します。送信されたユーザー名がエイリアス属性と一致する場合にのみ、`username` パラメータを返します。ユーザーが既存のパスワードを引き続き使用できるようにする場合は、Lambda レスポンスで属性 `finalUserStatus` を `CONFIRMED` に設定します。アプリケーションは、[ユーザー移行の Lambda トリガーのパラメータ](#) に示されるすべての "response" パラメータを返す必要があります。

⚠ Important

リクエストイベントオブジェクト全体をユーザー移行 Lambda コードに記録しないでください。このリクエストイベントオブジェクトには、ユーザーのパスワードが含まれます。ログをサニタイズしない場合、パスワードが CloudWatch Logs に表示されます。

5. Amazon Cognito はユーザープールにユーザープロフィールを作成し、トークンをアプリケーションクライアントに返します。
6. アプリケーションはトークンの取り込みを実行し、ユーザー認証を受け入れ、リクエストされたコンテンツに進みます。

ユーザーを移行したら、USER_SRP_AUTH を使用してサインインします。Secure Remote Password (SRP) プロトコルは、パスワードをネットワーク上に送信しないため、移行時に使用する USER_PASSWORD_AUTH フローよりもセキュリティ上の利点が大きくなります。

移行中にクライアントデバイスやネットワークの問題などのエラーが発生した場合、アプリケーションは Amazon Cognito ユーザープール API からエラーレスポンスを受け取ります。この場合 Amazon Cognito はユーザープールにユーザーアカウントを作成しない可能性があります。その後、ユーザーは再度サインインを試行する必要があります。サインインが繰り返し失敗した場合、アプリケーションでパスワードを忘れた場合のフローを使用してユーザーのパスワードをリセットする必要があります。

パスワードを忘れた場合のフローでは、UserMigration_ForgotPassword イベントソースを使用して [ユーザー移行の Lambda トリガー](#) 関数も呼び出します。ユーザーはパスワードのリセットをリクエストしてもパスワードを送信しないため、Amazon Cognito は Lambda 関数に送信するイベントにパスワードを含めません。関数では、既存のユーザーディレクトリ内のユーザーのみを検索し、ユーザープールのユーザープロフィールに追加する属性を返すことができます。関数が呼び出しを完了して Amazon Cognito に応答を返すと、ユーザープールはパスワードリセットコードを E メールまたは SMS で送信します。アプリケーションで、ユーザーに確認コードと新しいパスワードの入力を促し、その情報を [ConfirmForgotPassword](#) API リクエストで Amazon Cognito に送信します。。マネージドログインでは、[パスワードを忘れた場合] フロー用の組み込みページを使用することもできます。

その他のリソース

- [Amazon Cognito ユーザープールへのユーザー移行方法](#)

CSV ファイルからユーザープールへのユーザーのインポート

外部 ID ストアがあり、かつ、新しいローカルユーザーのためにユーザープールを準備する時間がある場合、カンマ区切り値 (CSV) ファイルからの一括ユーザーインポートは、Amazon Cognito ユーザープールへの移行のための手間とコストが少ないオプションになります。CSV ファイルインポートは、テンプレートファイルをダウンロードして入力し、インポートジョブのユーザープールにファイルを渡すプロセスです。CSV インポートを使用して、テストユーザーをすばやく作成できます。また、外部 ID ストアへの読み取り API リクエストをプログラムでファイルに入力したうえで、その詳細と属性を解析してファイルへの書き込みオペレーションにすることもできます。

インポートプロセスは、[パスワード] 以外のすべてのユーザー属性の値を設定します。セキュリティのベストプラクティスでは、パスワードはプレーンテキスト形式として使用できない必要があり、ハッシュのインポートはサポートされていないため、パスワードのインポートはサポートされません。つまり、ユーザーは最初にサインインした際にパスワードを変更する必要があります。このメソッドを使用してインポートすると、ユーザーは RESET_REQUIRED 状態になります。

CSV からユーザーをインポートする最も簡単な方法は、ユーザープールで [\[パスワードなしのサインイン\]](#) を有効にすることです。E メールアドレスと電話番号の属性、さらにユーザープールの適切な設定により、ユーザーはインポートジョブの完了直後に E メールまたは SMS ワンタイムパスワード (OTP) を使用してサインインできます。詳細については、「[インポートされたユーザーにパスワードをリセットするように要求](#)」を参照してください。

[AdminSetUserPassword](#) API リクエストで Permanent パラメータを true に設定することで、ユーザーのパスワードを設定することもできます。CSV インポートは、ユーザープール内で請求対象の月間アクティブユーザー (MAU) にはカウントされません。ただし、パスワードリセットオペレーションは MAU を発生させます。パスワードを使用する多数のユーザーをインポートしてもすぐにはアクティブにならない場合、コストを管理するには、ユーザーがサインインして RESET_REQUIRED チャレンジを受け取ったときに、新しいパスワードの入力を求めるようにアプリケーションを設定します。

Note

各ユーザーの作成日は、ユーザーがユーザープールにインポートされた時間です。作成日は、インポートされた属性の 1 つではありません。

ユーザーインポートジョブを作成する手順

1. AWS Identity and Access Management (IAM) コンソールで Amazon CloudWatch Logs ロールを作成します。
2. ユーザーインポート .csv ファイルを作成します。
3. ユーザーインポートジョブを作成し、実行します。
4. ユーザーインポート .csv ファイルをアップロードします。
5. ユーザーインポートジョブを起動し、実行します。
6. CloudWatch を使用してイベントログを確認します。
7. インポートされたユーザーがパスワードをリセットするように要求します。

その他のリソース

- ユーザープール間でのユーザーアカウントのエクスポートに関する「[Cognito User Profiles Export リファレンスアーキテクチャ](#)」

トピック

- [CloudWatch Logs IAM ロールの作成](#)
- [ユーザーインポート CSV ファイルの作成](#)
- [Amazon Cognito ユーザープールのインポートジョブの作成と実行](#)
- [ユーザープールのインポート結果を CloudWatch コンソールに表示](#)
- [インポートされたユーザーにパスワードをリセットするように要求](#)

CloudWatch Logs IAM ロールの作成

Amazon Cognito の CLI または API を使用している場合は、CloudWatch IAM ロールを作成する必要があります。以下の手順では、Amazon Cognito がインポートジョブの結果を CloudWatch Logs に書き込むために使用できる IAM ロールを作成する方法について説明します。

Note

Amazon Cognito コンソールでインポート ジョブを作成すると、同時に IAM ロールを作成できます。新しい IAM ロールを作成することを選択すると、Amazon Cognito は適切な信頼ポリシーと IAM ポリシーをロールに自動的に適用します。

ユーザープールインポート用の CloudWatch Logs IAM ロールを作成するには (AWS CLI、API)

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/iam/> で IAM コンソールを開きます。
2. の新しい IAM ロールを作成します AWS のサービス。詳しい手順については、「AWS Identity and Access Management ユーザーガイド」の「[AWS のサービス用ロールの作成](#)」を参照してください。
 - a. 信頼されたエンティティタイプのユースケースを選択するときは、任意のサービスを選択してください。Amazon Cognito は現在、サービスのユースケースに記載されていません。
 - b. [Add permissions] (アクセス許可の追加) 画面で、[Create policy] (ポリシーの作成) を選択し、次のポリシーステートメントを挿入します。**REGION** を、などのユーザープール AWS リージョン の に置き換えます us-east-1。 **ACCOUNT** を AWS アカウント ID に置き換えます (111122223333 など)。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:111122223333:log-group:/aws/cognito/*"
      ]
    }
  ]
}
```

3. ロールを作成したときに Amazon Cognito を信頼されたエンティティとして選択しなかったため、ロールの信頼関係を手動で編集する必要があります。IAM コンソールのナビゲーションページから [Roles] (ロール) を選択し、作成した新しいロールを選択します。
4. [信頼関係] タブを選択します。

5. [Edit trust policy] (信頼ポリシーを編集) を選択します。
6. 次のポリシーステートメントを [Edit trust policy] (信頼ポリシーを編集) に貼り付け、既存のテキストを置き換えます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "cognito-idp.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

7. [ポリシーの更新] を選択してください。
8. ロールの ARN を書き留めておきます。インポートジョブを作成するときに、ARN を指定します。

ユーザーインポート CSV ファイルの作成

既存のユーザーをユーザープールにインポートする前に、インポートするユーザーとその属性を含むカンマ区切り値 (CSV) ファイルを作成する必要があります。ユーザープールから、ユーザープールの属性スキーマを反映するヘッダーを含むユーザーインポートファイルを取得できます。その後、[CSV ファイルのフォーマット](#) のフォーマット要件に一致するユーザー情報を挿入できます。

CSV ファイルヘッダーのダウンロード (コンソール)

CSV ヘッダーファイルをダウンロードするには、次の手順に従います。

CSV ファイルヘッダーをダウンロードするには

1. [Amazon Cognito コンソール](#) に移動します。AWS 認証情報の入力を求められる場合があります。
2. [User Pools] (ユーザープール) を選択します。

3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択します。
5. [Import users] (ユーザーのインポート) セクションで、[Create an import job] (インポートジョブの作成) を選択します。
6. [Upload CSV] (CSV のアップロード) で [template.csv] リンクを選択し、CSV ファイルをダウンロードします。

CSV ファイルヘッダーのダウンロード (AWS CLI)

正しいヘッダーのリストを取得するには、[ユーザー] メニューの [ユーザーをインポート] で、[インポートジョブを作成] を選択します。表示されるダイアログで `template.csv` リンクを選択し、ユーザープール属性を含むテンプレートファイルをダウンロードします。

次の CLI コマンドを実行することもできます。`USER_POOL_ID` は、ユーザーをインポートする先のユーザープールのユーザープール識別子です。

```
aws cognito-idp get-csv-header --user-pool-id "USER_POOL_ID"
```

レスポンス例:

```
{
  "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"
}
```

CSV ファイルのフォーマット

ダウンロードしたユーザーインポート CSV ヘッダーファイルは次の文字列のようになります。ユーザープールに追加したカスタム属性も含まれます。

```
cognito:username,name,given_name,family_name,middle_name,nickname,preferred_username,profile,pi
```

ユーザーのためのこのヘッダーと属性値が含まれ、次のルールに従ってフォーマットされているように CSV ファイルを編集します。

Note

電話番号の適切な形式など属性値の詳細については、「[ユーザー属性の操作](#)」を参照してください。

- ファイルの最初の行はユーザー属性の名前を含む、ダウンロードされたヘッダーの行です。
- CSV ファイルの行の順序は重要ではありません。
- 最初の行の後の各行に、ユーザーの属性値が含まれます。
- ヘッダーのすべての列が存在している必要がありますが、列の値を指定する必要はありません。
- 以下の属性は必須です:
 - cognito:username
 - [email_verified] または [phone_number_verified]
 - 自動確認された属性の少なくとも一つは、各ユーザーに対して true である必要があります。自動検証属性は、新しいユーザーがユーザープールに参加したときに Amazon Cognito が自動的にコードを送信する E メールアドレスまたは電話番号です。
 - ユーザープールは [email_verified] または [phone_number_verified] の少なくとも一つの自動確認された属性が必要です。ユーザープールに自動確認された属性が存在しない場合は、インポートジョブは開始されません。

- ユーザープールに一つの自動確認された属性がある場合のみ、その属性はユーザーごとに確認する必要があります。たとえば、ユーザープールに自動確認された属性として [phone_number] だけがある場合、[phone_number_verified] 値はそれぞれのユーザーに対して true である必要があります。

 Note

ユーザーがパスワードをリセットするには、検証済みの E メールまたは電話番号が必要です。Amazon Cognito は、CSV ファイルで指定された E メールアドレスまたは電話番号にパスワードのリセットコードが含まれるメッセージを送信します。メッセージが電話番号に送信される場合は、SMS メッセージで送信されます。詳細については、「[サインアップ時に連絡先情報を検証する](#)」を参照してください。

- [email] ([email_verified] が true の場合)
- [phone_number] ([phone_number] が true の場合)
- ユーザープールを作成する際に必須と指定した任意の属性
- 文字列である属性値には、引用符は使用できません。
- 属性値にカンマが含まれる場合、カンマの前にバックスラッシュ (\) を置く必要があります。これは、CSV ファイルのフィールドがカンマで区切られているためです。
- CSV ファイルのコンテンツは、バイトオーダーマークのない UTF-8 形式にする必要があります。
- [cognito:username] フィールドは必須であり、ユーザープール内で一意である必要があります。任意の Unicode 文字列を使えます。ただし、スペースまたはタブを含めることはできません。
- [birthdate] (生年月日) の値がある場合、「**mm/dd/yyyy**」の形式である必要があります。たとえば、生年月日が 1985 年 2 月 1 日だとすると、**02/01/1985** にエンコードされる必要があります。
- cognito:mfa_enabled フィールドは、ユーザープールの MFA 要件に対応している必要があります。ユーザープールで多要素認証 (MFA) を必須に設定している場合、このフィールドはすべてのユーザーに対して true または空白にする必要があります。MFA をオフに設定している場合、このフィールドはすべてのユーザーに対して false にする必要があります。空白の値により、インポートしたユーザーの MFA 対応ステータスは、ユーザープールが要求する状態に設定されます。cognito:mfa_enabled 値を設定したかどうかにかかわらず、有効な MFA 要素なしで、MFA を必須とするユーザープールにユーザーをインポートできます。この状態のユーザーは MFA がアクティブですが、E メール属性、電話番号属性、または TOTP を、ユーザープールの有効な MFA 要素として設定するまでサインインできません。
- 最大行数は 16,000 文字です。

- CSV ファイルの最大サイズは 100 MB です。
- ファイルの最大行数 (ユーザー) は 500,000 です。この最大値にはヘッダー行は含まれません。
- [updated_at] フィールド値はエポック時間 (秒) であると想定します。たとえば **1471453471** などです。
- 属性値の先頭または末尾の空白は切り捨てられます。

次のリストは、カスタム属性のないユーザープールの CSV インポートファイルの例です。ユーザープールスキーマは、この例とは異なる場合があります。その場合は、ユーザープールからダウンロードした CSV テンプレートにテスト値を入力する必要があります。

[illegible]

Amazon Cognito ユーザープールのインポートジョブの作成と実行

このセクションでは、Amazon Cognito コンソールと AWS Command Line Interface () を使用してユーザープールのインポートジョブを作成して実行する方法について説明しますAWS CLI。

トピック

- [CSV ファイルからのユーザーのインポート \(コンソール\)](#)
- [ユーザーのインポート \(AWS CLI\)](#)

CSV ファイルからのユーザーのインポート (コンソール)

次の手順は、CSV ファイルからユーザーをインポートする方法について説明します。

CSV ファイルからユーザーをインポートするには (コンソール)

1. [Amazon Cognito コンソール](#)に移動します。AWS 認証情報の入力を求められる場合があります。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [ユーザー] メニューを選択します。

5. [Import users] (ユーザーのインポート) セクションで、[Create an import job] (インポートジョブの作成) を選択します。
6. [Create import job] (インポートジョブの作成) ページで、ジョブ名を入力します。
7. [Create a new IAM role] (新しい IAM ロールの作成)、または[Use an existing IAM role] (既存のロールを使用する) を選択します。
 - a. [Create a new IAM role] (新しい IAM ロールの作成) を選択した場合は、新しいロールの名前を入力します。Amazon Cognito は、正しいアクセス許可と信頼関係を持つロールを自動的に作成します。インポートジョブを作成する IAM プリンシパルに、IAM ロールを作成するアクセス許可が必要です。
 - b. [Use an existing IAM role] (既存の IAM ロールを使用する) を選択した場合は、[IAM role selection] (IAM ロール選択) のリストからロールを選択します。このロールには、[CloudWatch Logs IAM ロールの作成](#) で説明されているアクセス許可と信頼ポリシーが必要です。
8. [CSV をアップロード] で [ファイルを選択] を選択し、準備した CSV ファイルをアタッチします。
9. [Create job] (ジョブの作成) を選択してジョブを送信しますが、後で開始します。[Create and start job] (ジョブを作成して開始) を選択してジョブを送信し、すぐに開始します。
10. ジョブを作成したが開始しなかった場合は、後で開始できます。[ユーザー] メニューの [ユーザーをインポート] で、インポートジョブを選択し、[開始] を選択します。AWS SDK から [StartUserImportJob](#) API リクエストを送信することもできます。
11. ユーザーインポートジョブの進行状況は、[ユーザー] メニューの [ユーザーをインポート] でモニタリングします。ジョブが成功しない場合は、[Status] (ステータス) 値を選択できます。詳細については、[View the CloudWatch logs for more details] (詳細については CloudWatch ログを表示する) を選択して、CloudWatch ログコンソールで問題がないか確認してください。

ユーザーのインポート (AWS CLI)

ユーザープールにユーザーをインポートするには、次の CLI コマンドが使用できます:

- `create-user-import-job`
- `get-csv-header`
- `describe-user-import-job`
- `list-user-import-jobs`
- `start-user-import-job`

- stop-user-import-job

これらのコマンドに関するコマンドラインオプションのリストを取得するには、help コマンドラインオプションを使用します。次に例を示します。

```
aws cognito-idp get-csv-header help
```

ユーザーインポートジョブの作成

CSV ファイルを作成した後、次の CLI コマンドを使用してユーザーインポートジョブを作成します。`[JOB_NAME]` にはジョブに付ける名前を指定します。`[USER_POOL_ID]` には以前と同じユーザープール ID を指定します。`[ROLE_ARN]` には、[CloudWatch Logs IAM ロールの作成](#) で次のとおり受け取ったロール ARN を指定します。

```
aws cognito-idp create-user-import-job --job-name "JOB_NAME" --user-pool-id "USER_POOL_ID" --cloud-watch-logs-role-arn "ROLE_ARN"
```

レスポンスで返される `PRE_SIGNED_URL` は 15 分間有効です。この時間以後は無効になり、新規のユーザーインポートジョブを作成して新しい URL を取得し直す必要があります。

Example レスポンス:

```
{
  "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
  }
}
```

ユーザーインポートジョブのステータス値

ユーザーインポートのコマンドに対する応答には、Status を示す次のような値が見つかります。

- Created - ジョブが作成されましたが、開始前の状態です。
- Pending - 遷移状態です。ジョブを開始しましたが、まだユーザーのインポートは開始していません。
- InProgress - ジョブが開始され、ユーザーインポートが進行中です。
- Stopping - ジョブを停止する処理中です。ユーザーインポートは停止していません。
- Stopped - ジョブは停止中で、ユーザーインポートも停止しています。
- Succeeded - ジョブは正常に完了しました。
- Failed - エラーのためジョブは停止しました。
- Expired - ジョブを作成しましたが、24 ~ 48 時間以内にジョブを起動しませんでした。ジョブに関連付けられたすべてのデータは削除され、ジョブを開始することはできません。

CSV ファイルのアップロード

以下の curl コマンドを使用して、ユーザーデータが含まれる CSV ファイルを create-user-import-job コマンドのレスポンスから取得した事前署名済みの URL にアップロードします。

```
curl -v -T "PATH_TO_CSV_FILE" -H "x-amz-server-side-encryption:aws:kms"  
"PRE_SIGNED_URL"
```

このコマンドの出力から、次のようなメッセージを見つけます: "We are completely uploaded and fine" このメッセージは、ファイルが正常にアップロードされたことを示します。ユーザープールは、インポートジョブの実行後にインポートファイルに情報を保持しません。完了または期限切れになると、Amazon Cognito は、アップロードした CSV ファイルを削除します。

ユーザーインポートジョブの説明

ユーザーインポートジョブの説明を取得するには、次のコマンドを使用します。**USER_POOL_ID** はユーザープール ID です。**JOB_ID** は、ユーザーインポートジョブの作成時に返されたジョブ ID です。

```
aws cognito-idp describe-user-import-job --user-pool-id "USER_POOL_ID" --job-id  
"JOB_ID"
```

Example レスポンス例:

```
{
```

```
"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
}
```

前述の例の出力では、**PRE_SIGNED_URL** は CSV のアップロード先の URL です。**ROLE_ARN** は、ロールの作成時に受け取った CloudWatch Logs ロール ARN です。

ユーザーインポートジョブを表示する

ユーザーインポートジョブを一覧表示するには、次のコマンドを使用します:

```
aws cognito-idp list-user-import-jobs --user-pool-id "USER_POOL_ID" --max-results 2
```

Exampleレスポンス例:

```
{
  "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"
}

```

ジョブは時系列で、作成が最後から最初の順で表示されます。2 番目のジョブのあとの **PAGINATION_TOKEN** 文字列は、このリストコマンドについて、まだ結果が残っていることを示します。残りの結果を表示するには、次のように `--pagination-token` オプションを使用します:

```
aws cognito-idp list-user-import-jobs --user-pool-id "USER_POOL_ID" --max-results 10 --
pagination-token "PAGINATION_TOKEN"
```

ユーザーインポートジョブの開始

ユーザーインポートジョブを開始するには、次のコマンドを使用します:

```
aws cognito-idp start-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

インポートジョブは、アカウントごとに 1 つしかアクティブにできません。

Example レスポンス例:

```

{
    "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
  }
}
```

ユーザーインポートジョブの停止

ユーザーインポートジョブの進行中に停止するには、次のコマンドを使用します。ジョブを停止すると、再開することはできません。

```
aws cognito-idp stop-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

Exampleレスポンス例:

```
{
  "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
  }
}
```

ユーザープールのインポート結果を CloudWatch コンソールに表示

インポートジョブの結果は、Amazon CloudWatch コンソールで表示できます。

トピック

- [結果の表示](#)
- [結果の解釈](#)

結果の表示

次のステップでは、ユーザープールのインポートの結果を表示する方法を説明します。

ユーザープールのインポート結果を見るには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/cloudwatch/> で CloudWatch コンソールを開きます。
2. [ログ] を選択します。
3. ユーザープールのインポートジョブに使用するロググループを選択します。ロググループ名は /aws/cognito/userpools/**USER_POOL_ID**/**USER_POOL_NAME** の形式です。
4. 先ほど実行したユーザーインポートジョブのログを選択します。ログ名は **JOB_ID**/**JOB_NAME** の形式です。ログの結果は行番号でユーザーを示しています。ユーザーデータはログに書き込まれていません。各ユーザーに対して、次のような行が表示されます。
 - [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).

結果の解釈

正常にインポートしたユーザーは、ステータスが「PasswordReset」に設定済みです。

次の場合、ユーザーはインポートされませんが、インポートジョブは継続します。

- true に設定された自動検証された属性はありません。
- ユーザーデータはスキーマに一致しません。
- 内部エラーによりユーザーをインポートできませんでした。

以下の場合、インポートジョブは失敗します。

- Amazon CloudWatch Logs ロールを想定することができず、正しいアクセスポリシーがない、または削除されている。
- ユーザープールが削除されている。
- Amazon Cognito が .csv ファイルを解析できない。

インポートされたユーザーにパスワードをリセットするように要求

ユーザープールがパスワードベースのサインインのみを提供している場合、ユーザーはインポート後にパスワードをリセットする必要があります。初めてサインインするときは、任意のパスワードを入力できます。Amazon Cognito は、アプリケーションからのサインインリクエストに対する API レスポンスで、新しいパスワードを入力するようユーザーに求めます。

ユーザープールにパスワードなしの認証要素がある場合、Amazon Cognito はインポートしたユーザーに対してデフォルトでその認証要素を使用します。ユーザーは、新しいパスワードの入力を求められず、パスワードなしの E メールまたは SMS OTP ですぐにサインインできます。また、ユーザー名パスワードやパスキーなど、他のサインイン方法を完了できるように、ユーザーにパスワードの設定を求めることもできます。ユーザーインポート後のパスワードなしのサインインには、以下の条件が適用されます。

1. 使用可能なパスワードなしのサインイン要素に対応する属性を持つユーザーをインポートする必要があります。ユーザーが E メールアドレスでサインインできる場合は、email 属性をインポートする必要があります。電話番号の場合は、phone_number 属性をインポートする必要があります。両方の場合は、いずれかの属性の値をインポートします。
2. 通常、ユーザーはパスワードをリセットする必要がある RESET_REQUIRED 状態でインポートされます。パスワードなしの要素でサインインできる状態でインポートされた場合、Amazon Cognito はユーザーの状態を CONFIRMED に設定します。

パスワードなしの認証の設定方法や、その認証フローをアプリケーションで構築する方法などの詳細については、「[Amazon Cognito ユーザープールによる認証](#)」を参照してください。

次の手順では、CSV ファイルのインポート後、カスタム構築されたログインメカニズムにおける、RESET_REQUIRED のローカルユーザーのユーザーエクスペリエンスについて説明します。ユーザーがマネージドログインでサインインする場合は、[パスワードを忘れた場合] オプションを選択し、E メールまたはテキストメッセージに記載されているコードを入力してパスワードを設定するよう指示します。

インポートされたユーザーにパスワードをリセットするように要求

1. アプリケーションで、ランダムなパスワードを使用した InitiateAuth により、現在のユーザーのサインインをサイレントに試行します。
2. PreventUserExistenceErrors が有効な場合、Amazon Cognito は NotAuthorizedException を返します。そうでない場合は PasswordResetRequiredException を返します。

3. アプリケーションが ForgotPassword API リクエストを行い、ユーザーのパスワードをリセットします。
 - a. アプリケーションは ForgotPassword API リクエストでユーザー名を送信します。
 - b. Amazon Cognito は、確認済みの E メールまたは電話番号にコードを送信します。宛先は、CSV ファイルで email_verified と phone_number_verified に指定した値によって異なります。ForgotPassword リクエストへのレスポンスは、コードの宛先を示します。
-  **Note**

ユーザープールは、E メールまたは電話番号を確認するように設定する必要があります。詳細については、「[ユーザーアカウントのサインアップと確認](#)」を参照してください。
- c. アプリケーションは、コードがコードの送信先を確認するメッセージをユーザーに表示し、ユーザーにコードと新しいパスワードを入力するように求めます。
 - d. ユーザーがアプリでコードと新しいパスワードを入力します。
 - e. アプリケーションは ConfirmForgotPassword API リクエストでコードと新しいパスワードを送信します。
 - f. アプリケーションはユーザーをサインインにリダイレクトします。

ユーザー属性の操作

属性は、名前、E メールアドレス、電話番号など、個々のユーザーを識別できる各種情報です。新しいユーザープールにはデフォルトの標準属性のセットがあります。のユーザープール定義にカスタム属性を追加することもできます AWS マネジメントコンソール。このトピックでは、それらの属性について詳しく説明し、ユーザープールの設定のヒントを示します。

ユーザーに関するすべての情報を属性に保存しないでください。例えば、使用統計やゲームスコアなどの頻繁に変化するユーザーデータは、Amazon Cognito Sync または Amazon DynamoDB などの個別のデータストアに保存してください。

ユーザー属性文字列値の入力は、サニタイズしてから、ユーザープールに送信します。提案されたユーザー属性値を分析する 1 つの方法は、[サインアップ前](#)などの Lambda トリガーを使用することです。

 Note

ドキュメントや標準によっては、属性をメンバーと呼んでいます。

トピック

- [標準属性](#)
- [ユーザー名および優先ユーザー名](#)
- [ログイン属性のカスタマイズ](#)
- [カスタム属性](#)
- [属性の権限と範囲](#)

標準属性

Amazon Cognito は、[\[OpenID Connect の仕様\]](#) に基づいて、以下の標準属性セットをすべてのユーザーに割り当てます。標準属性値およびカスタム属性値は、デフォルトでは最大 2048 文字の任意の文字列ですが、一部の属性値には形式の制限があります。

標準属性は以下のとおりです。

- name
- family_name
- given_name
- middle_name
- nickname
- preferred_username
- profile
- picture
- website
- gender
- birthdate
- zoneinfo
- locale
- updated_at

- address
- email
- phone_number
- sub

sub を除き、標準属性はすべてのユーザーに対してデフォルトでオプションとなります。属性を必須にする場合は、ユーザープールの作成プロセスで、属性の横にある [必須] チェックボックスをオンにします。Amazon Cognito は、各ユーザーの sub 属性に一意のユーザー識別子の値を割り当てます。検証できるのは E メールと phone_number 属性だけです。

標準属性には、[DescribeUserPool API レスポンス](#)の SchemaAttributes パラメータで表示できる事前定義されたプロパティがあります。データ型、ミュータビリティ、長さの制約など、これらの属性プロパティのカスタム値を設定できます。標準属性プロパティを変更するには、[CreateUserPool スキーマパラメータ](#)でカスタム値を設定します。スキーマでは、必須の属性も設定できます。Amazon Cognito コンソールでユーザープールを作成するときに、標準属性のプロパティは変更できません。

Note

標準属性を必須とマークすると、ユーザーは、属性の値が指定されない限り登録することはできません。ユーザーを作成し、必須属性の値を指定しないようにするには、管理者は [AdminCreateUser](#) API を使用できます。ユーザープールを作成した後で、属性を必須と必須ではないの間に切り替えることができません。

標準属性の詳細と形式の制限

birthdate

値は、YYYY-MM-DD 形式で有効な 10 文字の日付にする必要があります。

E メール

ユーザーおよび管理者は、E メールアドレスの値を確認できます。

適切な AWS アカウント アクセス許可を持つ管理者は、ユーザーの E メールアドレスを変更し、検証済みとしてマークすることもできます。[AdminUpdateUserAttributes](#) API または [admin-update-user-attributes](#) AWS Command Line Interface (AWS CLI) コマンドを使用して、E メールアドレスを検証済みとしてマークします。このコマンドを使用すると、管理者

は `email_verified` 属性を `true` に変更できます。Amazon Cognito コンソールの [ユーザー] メニューでユーザーを編集して、E メールアドレスを検証済みとしてマークすることもできます。

値は、[有効な E メールアドレス文字列](#)とし、@ 記号とドメインを含む標準の E メール形式に従い、長さは 2,048 文字までで指定します。

phone_number

SMS 多要素認証 (MFA) がアクティブである場合、ユーザーは電話番号を入力する必要があります。詳細については、「[ユーザープールに MFA を追加します](#)」を参照してください。

ユーザーおよび管理者は、電話番号の値を確認できます。

適切な AWS アカウント アクセス許可を持つ管理者は、ユーザーの電話番号を変更し、検証済みとしてマークすることもできます。[AdminUpdateUserAttributes](#) API または [admin-update-user-attributes](#) AWS CLI コマンドを使用して、電話番号を検証済みとしてマークします。このコマンドを使用すると、管理者は `phone_number_verified` 属性を `true` に変更できます。Amazon Cognito コンソールの [ユーザー] メニューでユーザーを編集して、電話番号を検証済みとしてマークすることもできます。

Important

電話番号は次の形式に従う必要があります: 電話番号は、プラス記号 (+) から始めて国番号がその後に続く必要があります。電話番号には、プラス記号 (+) および数値のみを含めることができます。値をサービスに送信する前に、電話番号から括弧、スペース、ハイフン (-) などの他の文字を削除してください。例えば、米国の電話番号は **+14325551212** の形式に従っています。

preferred_username

`preferred_username` は、必要に応じて、またはエイリアスとして選択できますが、両方を選択することはできません。`preferred_username` がエイリアスである場合、ユーザーの確認後に [UpdateUserAttributes](#) API オペレーションをリクエストして属性値を追加できます。

sub

`sub` 属性に基づいてユーザーをインデックス化して検索します。`sub` 属性は、各ユーザープール内で一意のユーザー識別子です。ユーザーは `phone_number` や `email` などの属性を変更できます。`sub` 属性には固定値が含まれています。ユーザーの検索の詳細については、「[ユーザーアカウントの管理と検索](#)」を参照してください。

必須属性を表示する

以下の手順に従って、特定のユーザープールに必要な属性を表示します。

Note

ユーザープールを作成した後は、必須の属性を変更することはできません。

必須属性を表示する

1. の[Amazon Cognito](#)」に移動します AWS マネジメントコンソール。コンソールでプロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [サインアップ] メニューを選択します。
5. [Required attributes] (必須の属性) セクションで、ユーザープールの必須属性が表示されます。

ユーザー名および優先ユーザー名

username 値は個別の属性であり、name 属性とは異なります。各ユーザーには、username 属性があります。Amazon Cognito は、フェデレーティッドユーザーのユーザー名を自動的に生成します。Amazon Cognito ディレクトリにローカルユーザーを作成するには、username 属性を指定する必要があります。ユーザーを作成した後で、username 属性の値を変更することはできません。

デベロッパーは、preferred_username 属性を使用して、変更可能なユーザー名をユーザーに付与できます。詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

お客様のアプリケーションでユーザー名が不要の場合は、ユーザーにユーザー名を指定するように求める必要はありません。アプリでユーザー用の一意のユーザー名をバックグラウンドで作成できます。例えば、E メールアドレスとパスワードを登録しそれを使用してサインインするようにユーザーに求める場合は便利です。詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

username はユーザープール内で一意である必要があります。username は再利用できますが、削除されて使用されなくなった場合のみです。username 属性に対する文字列制約については、[SignUp](#) API リクエストの username プロパティを参照してください。

ログイン属性のカスタマイズ

ユーザープールを作成するときに、ユーザーが e メールアドレスまたは電話番号をユーザー名として使用してサインアップおよびサインインできるようにする場合は、ユーザー名属性を設定できます。また、エイリアス属性を設定してユーザーに選択肢を与えることもできます。サインアップするときに複数の属性を含めて、ユーザー名、優先ユーザー名、E メールアドレス、または電話番号でサインインできます。

Important

ユーザープールを作成した後は、この設定を変更することはできません。

エイリアス属性とユーザー名属性の選択方法

要件	エイリアス属性	ユーザー名属性
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 UsernameExistsException errors ³	Yes	No
Can assign the same email address or phone number attribute value to more than one user	Yes ⁴	No

¹ 使用可能なサインイン属性は、ユーザー名、E メールアドレス、電話番号、および優先ユーザー名です。

² E メールアドレスまたは電話番号を使用してサインインできます。

³ ユーザーが登録したメールアドレスや電話番号が重複している可能性があるが、ユーザー名がない場合、ユーザープールで `UsernameExistsException` エラーが生成されません。この動作はユーザー名存在エラーを防ぐとは無関係です。これはサインイン操作には適用されますが、サインアップ操作には適用されません。

⁴ 属性を確認した最後のユーザーのみが、その属性を使用してサインインできます。

オプション 1: 複数のログイン属性 (エイリアス属性)

ユーザーがユーザー名だけでなく別の属性でもサインインできる場合、その属性はエイリアスになります。ユーザーがサインインフォームのユーザー名フィールドでユーザー名と他の属性値から選択できるようにする場合は、エイリアスを設定します。username 属性は、ユーザーが変更できない固定値です。属性をエイリアスとしてマーキングすると、ユーザーはユーザー名の代わりにその属性を使用してサインインできます。E メールアドレス、電話番号、および任意ユーザー名属性は、エイリアスレコードとしてマーキングできます。例えば、E メールと電話をユーザープールのエイリアスとして選択すると、そのユーザープールのユーザーは、自分のユーザー名、E メールアドレス、または電話番号とパスワードを使ってサインインできます。

エイリアス属性を選択するには、ユーザープールを作成するときに [User name] (ユーザー名) と 1 つ以上の追加のサインインオプションを選択します。

Note

大文字と小文字を区別しないようにユーザープールを設定すると、ユーザーは小文字または大文字を使用してサインアップ、またはエイリアスを使用してサインインできるようになります。詳細については、Amazon Cognito API リファレンスの「[CreateUserPool](#)」を参照してください。

エイリアスとして E メールアドレスを選択した場合、Amazon Cognito は有効な E メールアドレス形式と一致するユーザー名を受け入れません。同様に、電話番号をエイリアスとして選択した場合、Amazon Cognito は、有効な電話番号形式に一致するユーザープールのユーザー名を受け入れません。

Note

エイリアス値は、ユーザープール内で一意にする必要があります。エイリアスがメールアドレスまたは電話番号に設定される場合、指定される値は 1 つのアカウントのみで検証状

態になります。サインアップ時に E メールアドレスまたは電話番号をエイリアス値として指定し、別のユーザーがそのエイリアス値を既に使用している場合、登録は成功します。ただし、ユーザーがこの E メール (または電話番号) を持つアカウントを確認して有効なコードを入力しようとする場合、Amazon Cognito は、AliasExistsException エラーを返します。エラーは、この E メールアドレス (または電話番号) を持つアカウントが既に存在していることを示します。この時点で、ユーザーは新しいアカウントの作成を中止して、古いアカウントのパスワードのリセットを試すことができます。ユーザーが新しいアカウントの作成を続行する場合は、アプリは forceAliasCreation オプションを使用して ConfirmSignUp API を呼び出す必要があります。forceAliasCreation を使用する ConfirmSignUp は、エイリアスを前のアカウントから新しく作成されたアカウントに移動し、前のアカウントで未確認の属性をマークします。

ユーザーが電話番号および E メールアドレスを確認した後、電話番号および E メールアドレスのみがユーザーのアクティブエイリアスになります。エイリアスとしてこれらを使用する場合は、E メールアドレスと電話番号の自動確認を選択することをお勧めします。

ユーザーがサインアップするときに、E メールアドレスおよび電話番号の属性の UsernameExistsException エラーを防ぐために、エイリアス属性を選択します。

preferred_username 属性をアクティブにして、username 属性値が変更されていないときに、ユーザーがサインインに使用するユーザー名を変更できるようにします。このユーザーエクスペリエンスをセットアップする場合は、新しい username 値を preferred_username として送信し、エイリアスとして preferred_username を選択します。こうすることで、ユーザーは入力した新しい値でサインインできます。preferred_username がエイリアスとして選択された場合、ユーザーはアカウントが確認されている場合にのみ、値を指定できます。登録時にはこの値は指定できません。

ユーザーがユーザー名を使用してサインアップする場合、ユーザーが次の 1 つ以上のエイリアスを使用してサインインできるかどうかを選択できます。

- 検証済みの E メールアドレス
- 検証済みの電話番号
- 優先ユーザー名

これらのエイリアスはユーザーがサインアップした後に変更できます。

⚠ Important

ユーザープールがエイリアスによるサインインをサポートしている場合、ユーザーを認可または検索するときに、サインイン属性のいずれかを使用してユーザーを特定することは避けてください。固定値のユーザー識別子 `sub` は、ユーザーのアイデンティティを示す唯一の一貫した指標です。

エイリアスを使用してサインインできるように、ユーザープールを作成する場合は、次の手順を含めます。

Phone number or email address (console)

ユーザープールの作成時に、E メールアドレスと電話番号をエイリアス属性として設定する必要があります。

Amazon Cognito コンソールでユーザー名エイリアスを使用してユーザープールを作成するには

1. AWS マネジメントコンソールの [Amazon Cognito](#) に移動します。コンソールでプロンプトが表示されたら、AWS 認証情報を入力します。
2. [開始方法] ボタンまたは [ユーザープールを作成] ボタンを使用して、新しいユーザープールを作成します。
3. [アプリケーションを定義] でアプリケーション設定を選択します。
4. [サインイン識別子のオプション] の [オプションを設定] で、[ユーザー名]の横のチェックボックスをオンにします。さらに他のオプション ([E メール] と [電話番号]) を 1 つ以上選択します。
5. エイリアス属性を [サインアップのための必須属性] として選択します。Amazon Cognito は、マネージドログインのサインアップフォームで必須属性の値を指定するよう新規ユーザーに求めます。
6. マネージドログインのサインイン後に、[リターン URL を追加] で、アプリケーションからリダイレクトするためのコールバック URL を設定します。
7. [作成] を選択します。

Phone number or email address (API/SDK)

[CreateUserPool](#) API オペレーションを使用して新しいユーザープールを作成します。次のように `AliasAttributes` パラメータを設定します。電話番号エイリアスのみが必要な場合は、`email`

エントリを削除できます。E メールアドレスエイリアスのみが必要な場合は、phone_number エントリを削除できます。

```
"AliasAttributes": [  
  "email",  
  "phone_number"  
],
```

Preferred username (API/SDK)

Amazon Cognito コンソールは、エイリアスとして preferred_username を使用せずに、ユーザープールを作成します。エイリアスを使用してユーザープールを作成するには、AWS SDK preferred_username で [CreateUserPool](#) API リクエストを使用してユーザープールを設定します。サインアップ時に任意のユーザー名属性を作成できるようにするには、preferred_username を必須属性として設定します。Amazon Cognito は、マネージドログインのサインアップフォームで必須属性の値を指定するよう新規ユーザーに求めます。Amazon Cognito コンソールでは preferred_username を必須属性として設定することは可能ですが、エイリアスとして使用することはできません。

エイリアスとして設定する

次に示すように、CreateUserPool リクエストの AliasAttributes パラメータで、preferred_username をエイリアスとして設定します。エイリアス属性として使用しない値は、リストから削除します。

```
"AliasAttributes": [  
  "email",  
  "phone_number",  
  "preferred_username"  
],
```

必須として設定する

Amazon Cognito は、マネージドログインのサインアップフォームで必須属性の値を指定するよう新規ユーザーに求めます。[CreateUserPool](#) リクエストの SchemaAttributes パラメータで、preferred_username を必須として設定します。

任意のユーザー名を必須属性として設定するには、次に示すように設定します。次の例では、preferred_username のデフォルトスキーマを変更して、必須として設定しています。AttributeDataType (デフォルトは string) や StringAttributeConstraints (デフォルトは 1~99 文字) など、他のスキーマパラメータは、デフォルト値を使用します。

```
"Schema": [  
  {  
    "Name": "preferred_username",  
    "Required": true  
  }  
]
```

オプション 2: サインイン属性 (ユーザー名属性) としての E メールアドレスまたは電話番号

ユーザー名として E メールアドレスまたは電話番号でユーザーがサインアップする場合、E メールアドレスのみ、電話番号のみ、またはいずれか一方でもサインアップできるようにするかを選択することができます。

ユーザー名属性を選択するには、ユーザープールの作成時に、[ユーザー名] をサインインオプションとして選択しないでください。

E メールまたは電話番号は一意である必要があり、別のユーザーによって既に使用されてはなりません。これを検証する必要はありません。ユーザーが E メールアドレスまたは電話番号を使用してサインアップした後、ユーザーは同じ E メールアドレスまたは電話番号を使用して新しいアカウントを作成することはできません。ユーザーは既存のアカウントを (必要に応じてパスワードをリセットして) 再利用できるだけです。ただし、ユーザーは E メールアドレスまたは電話番号を新しい E メールアドレスまたは電話番号に変更できます。E メールアドレスまたは電話番号が既に使用中ではない場合は、それが新しいユーザー名となります。

E メールアドレスと電話番号の両方をユーザー名属性として選択した場合、ユーザーは両方の属性値を指定できますが、サインインに使用できるのはいずれか一方のみです。サインインユーザー名は、[SignUp](#) の Username パラメータで渡した値に基づいて決まります。

Note

ユーザーが E メールアドレスをユーザー名として使用してサインアップした場合、ユーザー名を別の E メールアドレスに変更できますが、電話番号に変更することはできません。電話番号でサインアップする場合は、ユーザー名を別の電話番号を変更できますが、E メールアドレスに変更することはできません。

次のステップを使用して、ユーザープール作成のプロセス中に、E メールアドレスまたは電話番号を使用したサインアップおよびサインインをセットアップします。

Username attributes (console)

次の手順では、E メールアドレスまたは電話番号をユーザー名属性とするユーザープールを作成します。Amazon Cognito コンソールでのユーザー名属性のプロセスの違いは、ユーザー名をサインイン属性として設定しないことです。

Amazon Cognito コンソールでユーザー名属性を持つユーザープールを作成するには

1. AWS マネジメントコンソールの [Amazon Cognito](#) に移動します。コンソールでプロンプトが表示されたら、AWS 認証情報を入力します。
2. [開始方法] ボタンまたは [ユーザープールを作成] ボタンを使用して、新しいユーザープールを作成します。
3. [アプリケーションを定義] でアプリケーション設定を選択します。
4. [サインイン識別子のオプション] の [オプションを設定] で、ユーザー名属性として [E メール]、[電話番号]、または両方を選択します。[ユーザー名] は、オフのままにします。
5. ベストプラクティスとして、ユーザー名属性を [サインアップのための必須属性] として選択します。Amazon Cognito は、マネージドログインのサインアップフォームで必須属性の値を指定するよう新規ユーザーに求めます。ユーザー名属性を必須として設定しない場合、Amazon Cognito は新規ユーザーに属性値の入力を求めません。この場合は、ユーザーがサインインする前に、各ユーザーの E メールアドレスまたは電話番号を収集して送信するようにアプリケーションを設定する必要があります。
6. マネージドログインのサインイン後に、[リターン URL を追加] で、アプリケーションからリダイレクトするためのコールバック URL を設定します。
7. [作成] を選択します。

Username attributes (API/SDK)

[CreateUserPool](#) リクエストで、次に示すように UsernameAttributes パラメータを設定します。E メールアドレスのみをユーザー名としてサインインすることを許可するには、このリストで email のみを指定します。電話番号のみをユーザー名としてサインインすることを許可するには、phone_number のみを指定します。このパラメータは、サインインオプションとしてのユーザー名をオーバーライドします。

```
"UsernameAttributes": [  
    "email",  
    "phone_number"  
],
```

ユーザー名属性を設定すると、[SignUp](#) API リクエストにより、username パラメータで E メールアドレスまたは電話番号を渡すことができます。ユーザー名属性を使用したコード SignUp API オペレーションの動作は、以下のとおりです。

- username 文字列が有効な E メールアドレス形式 (例: user@example.com) の場合、ユーザープールはユーザーの email 属性の値として、自動的に username を入力します。
- username 文字列が有効な電話番号形式 (例: +12065551212) の場合、ユーザープールはユーザーの phone_number 属性の値として、自動的に username を入力します。
- username 文字列形式が E メールアドレスまたは電話番号の形式ではない場合は、SignUp API から例外が返されます。
- username 文字列に含まれている E メールアドレスまたは電話番号が既に使用されている場合、SignUp API によって例外が返されます。
- SignUp API は、username 属性にユーザーの [UUID](#) を入力します。この UUID は、ユーザー ID トークン内の sub クレームと同じ値です。

[ListUsers](#) オペレーションを除くすべての API で、E メールアドレスまたは電話番号をユーザー名の代わりに使用できます。ListUsers API リクエストでは、email または phone_number の Filter を指定できます。username でフィルタリングする場合は、E メールアドレスや電話番号ではなく、UUID ユーザー名を指定する必要があります。

カスタム属性

ユーザープールには、カスタム属性を最大 50 まで追加できます。カスタム属性の最小長または最大長を指定できます。ただし、すべてのカスタム属性の最大長は 2048 文字以下にする必要があります。カスタム属性の名前は、[SchemaAttributeType](#) の Name パラメータで使用した正規表現パターンと一致する必要があります。

各カスタム属性には以下の特徴があります。

- 文字列、数値、ブール値、または DateTime オブジェクトとして定義できます。Amazon Cognito は ID トークンにカスタム属性値を文字列としてのみ書き込みます。

Note

Amazon Cognito コンソールでは、文字列と数値をデータ型とするカスタム属性のみを追加できます。ブール値や DateTime 属性など、他のデータ型のオプション

は、[CreateUserPool](#) および [UpdateUserPool](#) API リクエストの SchemaAttributes プロパティでのみ使用できます。

- ユーザーが属性の値を提供することを要求することはできません。
- ユーザープールに追加した後で、削除または変更することはできません。
- 属性名の文字長は、Amazon Cognito が受け入れる制限内です。詳細については、「[Amazon Cognito のクォータ](#)」を参照してください。
- 変更可能またはイミュータブルとすることができます。ユーザーを作成するときに、イミュータブル属性に値を書き込むことができます。アプリケーションがその属性に対する書き込み権限を持っている場合、変更可能な属性の値を変更できます。詳細については「[属性の権限と範囲](#)」を参照してください。

 Note

コードおよび [ルールベースアクセスコントロールの使用](#) のルールの設定で、カスタム属性は標準属性と区別するために、custom: プレフィックスを必要とします。

また、[CreateUserPool](#) の SchemaAttributes プロパティで、ユーザープールを作成する際に、開発者属性を追加することもできます。開発者属性には dev: プレフィックスがあります。ユーザーのデベロッパー属性は、AWS 認証情報でのみ変更できます。開発者属性はレガシー機能であり、Amazon Cognito がアプリケーションクライアントの読み取り/書き込み権限と置き換えました。

以下の手順を使用して、新しくカスタム属性を作成します。

コンソールを使用してカスタム属性を追加する

1. の[Amazon Cognito](#)」に移動します AWS マネジメントコンソール。コンソールでプロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [サインアップ] メニューを選択し、[カスタム属性] セクションで [カスタム属性を追加] を選択します。
5. [Add custom attributes] (カスタム属性を追加する) のページで、新しい属性に関する以下の詳細情報を入力します。

- 名前を入力します。
- [Type] (タイプ) (文字列または数値) を選択します。
- 最小文字列の長さまたは数値を入力します。
- 最大文字列の長さまたは数値を入力します。
- ユーザーが初期値を設定した後で、カスタム属性の値を変更する権限をユーザーに付与する場合、[Mutable] (変更可能) を選択します。

6. [Save changes] (変更の保存) をクリックします。

属性の権限と範囲

各アプリケーションで、各ユーザー属性に読み取りと書き込み権限を設定できます。これにより、ユーザー用に保存する各属性の読み取りおよび変更に必要なアプリによるアクセスを制御できます。例えば、ユーザーが有料サービスを利用しているお客様であるかどうかを示すカスタム属性があるとします。アプリでこの属性は表示できますが、直接変更することはできません。代わりに、管理ツールまたはバックグラウンドプロセスを使用してこの属性を更新します。ユーザー属性の権限は、Amazon Cognito コンソール、Amazon Cognito API、または AWS CLI から設定できます。デフォルトでは、新しいカスタム属性を利用する前に、読み取り/書き込みの許可を設定する必要があります。デフォルトでは、アプリケーションクライアントを作成すると、すべての標準属性とカスタム属性に対する読み取り/書き込み許可がアプリケーションに付与されます。アプリケーションが必要とする情報量のみに制限するには、アプリケーションクライアント設定の属性に特定のアクセス権限を割り当てます。

ベストプラクティスとして、アプリケーションクライアントを作成するときに属性の読み取り/書き込みアクセス許可を指定します。アプリケーションのオペレーションに必要な最小限のユーザー属性セットへのアクセスをアプリケーションクライアントに付与します。

Note

デフォルト以外のアプリケーションクライアント権限を設定した場合、[UserPoolClient](#) は、ReadAttributes と WriteAttributes のみを返します。

属性権限を更新するには (AWS マネジメントコンソール)

1. の[Amazon Cognito](#)」に移動します AWS マネジメントコンソール。コンソールでプロンプトが表示されたら、AWS 認証情報を入力します。

2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [アプリケーションクライアント] メニューを選択し、リストからアプリケーションクライアントを選択します。
5. [属性権限] タブで、[編集] を選択します。
6. [Edit attribute read and write permissions] (属性の読み込みおよび書き込みアクセス権限を編集) ページで、読み取りおよび書き込み権限を設定し、[Save changes] (変更の保存) を選択します。

カスタム属性を使用して、アプリクライアントごとに以上の手順を繰り返します。

アプリケーションごとに、属性を読み取り可能または書き込み可能とマークできます。これは、標準属性とカスタム属性の両方に当てはまります。アプリは、読み取り可能としてマークした属性の値を取得したり、書き込み可能としてマークした属性の値を設定または変更したりできます。書き込み権限のない属性の値をアプリが設定しようとする、Amazon Cognito が `NotAuthorizedException` を返します。[GetUser](#) リクエストにはアプリケーションクライアントクレーム付きのアクセストークンが含まれます。Amazon Cognito は、アプリケーションクライアントが読み取ることができる属性の値のみを返します。アプリからのユーザーの ID トークンには、読み取り可能な属性に対応するクレームのみが含まれます。すべてのアプリケーションクライアントは、ユーザープールの必須属性を書き込むことができます。Amazon Cognito ユーザープール API リクエストで属性の値を設定できるのは、まだ値が設定されていない必須属性の値も指定する場合のみに限られます。

カスタム属性の読み取り書き込みアクセス許可にはそれぞれ異なる機能があります。ユーザープールの変更可能な属性または変更不可能な属性として作成でき、どのアプリクライアントでも読み取りまたは書き込み属性として設定できます。

変更不可能なカスタム属性は、ユーザー作成時に 1 回だけ更新できます。変更不可能な属性には以下のメソッドを入力できます。

- **SignUp:** ユーザーは、変更不可能なカスタム属性への書き込みアクセス許可を持つアプリクライアントにサインアップします。その属性の値を提供します。
- **サードパーティ IdP でのサインイン:** ユーザーは、変更不可能なカスタム属性への書き込みアクセス許可を持つアプリクライアントにサインインします。IdP のユーザープール設定には、提供されたクレームを変更不可能な属性にマップするルールがあります。これは可能ですが、ユーザーがサインインできるのは一度だけであるため、実用的ではありません。最初のサインイン試行の後

に 2 度目以降のサインイン試行がなされると、現在書き込み不可の属性へのマッピングルールにより、Amazon Cognito は、その試行を拒否します。

- AdminCreateUser: 変更不可能な属性の値を指定します。

スコープを使用した属性アクセス許可

AWS SDK または CDK、REST API、または で設定したユーザープールでは AWS CLI、OIDC スコープ を使用してアプリケーションクライアントの読み取りまたは書き込みアクセスを設定できます。oidc:profile。oidc:profile は、次の標準属性への読み取り/書き込みアクセスを許可します。

- name
- family_name
- given_name
- middle_name
- nickname
- preferred_username
- profile
- picture
- website
- gender
- birthdate
- zoneinfo
- locale

このリストは、[OIDC 仕様のセクション 2.4](#) で定義されているように、OIDC 標準属性から email、phone_number、sub、address を除外したものです。アプリケーションクライアントに割り当てることができるスコープについては、「[スコープ、M2M、リソースサーバー](#)」を参照してください。

oidc:profile スコープ内の属性に書き込むようにアプリケーションクライアントを設定するには、[CreateUserPoolClient](#) API リクエストまたは [UpdateUserPoolClient](#) API リクエストで、[WriteAttributes](#) の値を oidc:profile に設定します。他の属性も変更することをアプリケー

セッションに許可する場合は、その属性も設定します。同様に、これらの属性への読み取りアクセスを許可するには、`oidc:profile` を [ReadAttributes](#) の値に追加します。

ユーザープールを作成した後、属性の権限と範囲を変更できます。

ユーザープール JSON ウェブトークン (JWT) の理解

トークンは、アプリケーションが OIDC 認証の証明として使用し、リソースへのアクセスをリクエストできる認証のアーティファクトです。トークンのクレームはユーザーに関する情報です。ID トークンには、ユーザー名、苗字、E メールアドレスなど、身元に関するクレームが含まれています。アクセストークンには、認証されたユーザーがサードパーティー API、Amazon Cognito ユーザーのセルフサービス API オペレーション、および [userInfo エンドポイント](#) にアクセスできる `scope` のようなクレームが含まれています。アクセストークンと ID トークンの両方に、ユーザープール内でのユーザーのグループメンバーシップを示す `cognito:groups` クレームが含まれています。ユーザープールグループの詳細については、「[ユーザープールにグループを追加する](#)」を参照してください。

Amazon Cognito には、新しいトークンの取得や既存のトークンの取り消しに使用できる更新トークンもあります。[トークンを更新](#)して、新しい ID とアクセストークンを取得します。[トークンを取り消し](#)て、更新トークンによって許可されているユーザーアクセスを取り消します。

Amazon Cognito は、[base64url](#) でエンコードされた文字列としてトークンを発行します 任意の Amazon Cognito ID またはアクセストークンを `base64url` からプレーンテキスト JSON にデコードできます。Amazon Cognito の更新トークンは暗号化されており、ユーザープールのユーザーと管理者に対して透過的ではなく、ユーザープールのみが読み取ることができます。

トークンを使用した認証

ユーザーがアプリケーションにサインインすると、Amazon Cognito がログイン情報を検証します。ログインが正常に行われると、Amazon Cognito がセッションを作成し、認証されたユーザーの ID トークン、アクセストークン、および更新トークンを返します。これらのトークンを使用して、ダウンストリームのリソースや Amazon API Gateway などの API へのアクセス権をユーザーに付与できます。または、これらのトークンを他の AWS のサービスにアクセスするための一時的な AWS 認証情報と交換することができます。



トークンの保存

アプリケーションでは、さまざまなサイズのトークンを保存する必要があります。トークンのサイズは、クレームの追加、エンコーディングアルゴリズムの変更、暗号化アルゴリズムの変更による理由（ただし、これらに限定されない）で変更される場合があります。ユーザープールでトークンの取り消しを有効にした場合も、Amazon Cognito が JSON ウェブトークンにクレームを追加するので、そのトークンのサイズは増加します。アクセストークンと ID トークンに対し、新たなクレーム `origin_jti` および `jti` が追加されます。トークンの取り消しの詳細については、「[トークンの取り消し](#)」を参照してください。

Important

ベストプラクティスとして、アプリケーションのコンテキストにおけるすべての転送中と保管中のトークンを保護します。トークンには、アプリケーションのユーザーに関する個人識別情報、およびユーザープールで使用するセキュリティモデルに関する情報を含めることができます。

トークンをカスタマイズする

Amazon Cognito からアプリに渡すアクセストークンと ID トークンをカスタマイズできます。[トークン生成前の Lambda トリガー](#)では、トークンクレームを追加、変更、および抑制できます。トークン生成前トリガーは、Amazon Cognito からデフォルトのクレームセットを送信する先の Lambda 関数です。クレームには、OAuth 2.0 スコープ、ユーザープールグループのメンバーシップ、ユーザー属性などが含まれます。この関数は、必要に応じてランタイムに変更を加え、更新したトークンクレームを Amazon Cognito に返すことができます。

バージョン 2 のイベントによるアクセストークンのカスタマイズには追加料金がかかります。詳細については、「[Amazon Cognito の料金](#)」を参照してください。

トピック

- [ID トークンの理解](#)
- [アクセストークンの理解](#)
- [更新トークン](#)
- [トークン取り消しによるユーザーセッションの終了](#)
- [JSON ウェブトークンの検証](#)
- [ユーザープールトークンの有効期限とキャッシュの管理](#)

ID トークンの理解

ID トークンとは、name、email、および phone_number など、認証されたユーザーのアイデンティティに関するクレームが含まれる、[JSON ウェブトークン \(JWT\)](#) です。この ID 情報はアプリケーション内で使用できます。ID トークンは、リソースサーバーまたはサーバーアプリケーションに対するユーザーの認証にも使用できます。Web API オペレーションを使用して、アプリケーション外で ID トークンを使用することも可能です。このような場合は、ID トークン内のクレームを信頼する前に、ID トークンの署名を検証する必要があります。「[JSON ウェブトークンの検証](#)」を参照してください。

ID トークンの有効期限は、5 分から 1 日までの任意の値に設定できます。この値は、アプリケーションのクライアントごとに設定できます。

Important

ユーザーがマネージドログインでサインインすると、Amazon Cognito は 1 時間有効なセッション Cookie を設定します。アプリケーションでマネージドログインを使用して認証し、アクセストークンと ID トークンに 1 時間未満の最小期間を指定すると、ユーザーは Cookie の有効期限が切れるまで有効なセッションを維持できます。1 時間のセッション中に有効期限が切れるトークンがある場合、ユーザーは再認証しなくてもトークンを更新できます。

ID トークンのヘッダー

ヘッダーには 2 つの情報 (キー ID: kid、アルゴリズム: alg) が含まれています。

```
{
  "kid" : "1234example=",
  "alg" : "RS256"
}
```

kid

キー ID。その値は、トークンの JSON Web 署名 (JWS) をセキュア化するために使用されたキーを示します。ユーザープール署名キー ID は jwks_uri エンドポイントで確認できます。

kid パラメータの詳細については、「[Key identifier \(kid\) header parameter](#)」を参照してください。

alg

Amazon Cognito が、アクセストークンをセキュア化するために使用した暗号化アルゴリズム。ユーザープールは、SHA-256 による RSA 署名である RS256 暗号化アルゴリズムを使用します。

alg パラメータの詳細については、「[Algorithm \(alg\) header parameter](#)」を参照してください。

ID トークンのデフォルトペイロード

これは ID トークンのサンプルペイロードです。認証されたユーザーに関するクレームが含まれています。OpenID Connect (OIDC) 標準クレームに関する詳細については、「[OIDC standard claims](#)」を参照してください。[トークン生成前の Lambda トリガー](#) を使用して独自の設計のクレームを追加できます。

```
<header>.{
  "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "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-eeeeeeeeeeee",
  "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-eeeeeeeeeeee",
"email": "my-test-user@example.com"
}
.<token signature>
```

sub

認証されたユーザーの固有識別子 ([UUID](#)) またはサブジェクト。ユーザー名はユーザープール内で一意ではない可能性があります。sub クレームは、特定のユーザーを識別する最良の方法です。

cognito:groups

ユーザーをメンバーとするユーザープールグループの名前の配列。グループは、アプリに提示する識別子として使用したり、アイデンティティプールへの優先 IAM ロールのリクエストの生成に使用したりできます。

cognito:preferred_role

ユーザーの最も優先度の高いユーザープールグループに関連付けた IAM ロールの ARN。ユーザープールがこのロールクレームを選択する方法の詳細については、「[グループへの優先順位の値の割り当て](#)」を参照してください。

iss

トークンを発行した ID プロバイダー。クレームは以下のような形式になります。

`https://cognito-idp.<Region>.amazonaws.com/<your user pool ID>`

cognito:username

ユーザープール内のユーザーのユーザー名。

nonce

nonce クレームは、OAuth 2.0 の authorize エンドポイントへのリクエストに追加するものと、名前が同じパラメータから取得されます。パラメータを追加すると、Amazon Cognito が発行する ID トークンに nonce クレームが含まれます。これは、リプレイ攻撃からの保護のため

めに使用できます。リクエストで nonce 値を指定せずにサードパーティー ID プロバイダーを介した認証を行う場合、Amazon Cognito はノンスを自動的に生成および検証した上で、その値を nonce クレームとして ID トークンに追加します。Amazon Cognito での nonce クレームの実装は、[OIDC 標準](#)に基づいています。

origin_jti

ユーザーの更新トークンに関連付けられたトークン失効識別子。Amazon Cognito は、[エンドポイントの取り消し](#) または [RevokeToken](#) API オペレーションを使用してユーザーのトークンが取り消されたかどうかを確認するときに、origin_jti クレームを参照します。トークンを取り消すと、Amazon Cognito は同じ origin_jti 値を持つすべてのアクセストークンと ID トークンを無効にします。

cognito:roles

ユーザーのグループに関連付けられた IAM ロールの名前の配列。各ユーザープールグループには、1 つの IAM ロールを関連付けることができます。この配列は、優先順位に関係なく、ユーザーグループのすべての IAM ロールを表します。詳細については、「[ユーザープールにグループを追加する](#)」を参照してください。

aud

ユーザーを認証したユーザープールアプリケーション。Amazon Cognito は、アクセストークン client_id クレームで同じ値をレンダリングします。

identities

ユーザーの identities 属性の内容。この属性には、フェデレーションサインインによって、または[フェデレーションユーザーをローカルプロファイルにリンクすることによって](#)ユーザーにリンクした各サードパーティー ID プロバイダープロファイルに関する情報が含まれます。この情報には、プロバイダー名、プロバイダーの固有 ID、およびその他のメタデータが含まれます。

token_use

トークンの本来の目的。ID トークンでは、その値は id です。

auth_time

ユーザーが認証を完了した認証時刻 (Unix の時間形式)。

exp

ユーザーのトークンの有効期限が切れる有効期限 (Unix の時間形式)。

iat

Amazon Cognito がユーザーのトークンを発行した発行時刻 (Unix の時間形式)。

jti

JWT の一意の識別子。

この ID トークンには、「[OIDC standard claims](#)」に定義されている、OIDC の標準クレームを含めることができます。また、ユーザープールで定義されたカスタム属性が、この ID トークンに含まれる場合もあります。Amazon Cognito は、属性タイプに関係なく、カスタム属性値を文字列として ID トークンに書き込みます。

Note

ユーザープールのカスタム属性には、常にプレフィックスとして `custom:` が付けられています。

ID トークンの署名

ID トークンの署名は、JWT トークンのヘッダーとペイロードに基づいて計算されます。アプリが受け取る ID トークンのクレームを受け入れる前に、トークンの署名を検証してください。詳細については、「JSON Web トークンの検証」を参照してください。[JSON ウェブトークンの検証](#)。

アクセストークンの理解

ユーザープールアクセストークンには、認証されたユーザーに関するクレーム、ユーザーのグループのリスト、およびスコープのリストが含まれます。アクセストークンの目的は、API オペレーションを承認することです。ユーザープールは、ユーザーのセルフサービスオペレーションを許可するアクセストークンを受け入れます。例えば、アクセストークンを使用して、ユーザーにユーザー属性を追加、変更、または削除するアクセス権を付与することができます。

ユーザープールに追加したカスタムスコープから派生した [OAuth 2.0 スコープ](#) をアクセストークンに含めると、ユーザーが API から情報を取得することを承認できます。例えば、Amazon API Gateway は、Amazon Cognito のアクセストークンによる認可をサポートします。REST API オーソライザーにユーザープールからの情報を入力することも、Amazon Cognito を HTTP API の JSON ウェブトークン (JWT) オーソライザーとして使用することもできます。カスタムスコープでアクセストークンを生成するには、ユーザープールの [パブリックエンドポイント](#) を通じてアクセストークンをリクエストする必要があります。

エッセンシャル[機能プラン](#)またはプラス機能プランでは、トークン生成前の Lambda トリガーを実装することで、ランタイムにアクセストークンにスコープを追加することもできます。詳細については、「[トークン生成前の Lambda トリガー](#)」を参照してください。

openid スコープを持つユーザーのアクセストークンは、ユーザーの属性に関する詳細情報を [userInfo エンドポイント](#) にリクエストするためのアクセス許可です。userInfo エンドポイントから得られる情報の量は、アクセストークン内のその他のスコープ (すべてのユーザーデータを求める profile や E メールアドレスを求める email など) によって異なります。

aws.cognito.signin.user.admin スコープを持つユーザーのアクセストークンは、ユーザー属性の読み取りと書き込み、認証要素の一覧表示、多要素認証 (MFA) の設定、記憶されたデバイスの管理を行うためのアクセス許可です。アクセストークンによってこのスコープに付与される、属性へのアクセス権のレベルは、アプリケーションクライアントに割り当てた、属性の読み取り/書き込みアクセス許可と一致します。

アクセストークンは、[JSON ウェブトークン \(JWT\)](#) です。アクセストークンのヘッダーは ID トークンと同じ構造になっていますが、Amazon Cognito は、ID トークンに署名するキーとは異なるキーでアクセストークンに署名します。アクセスキー ID (kid) クレームの値は、同じユーザーセッションの ID トークンの kid クレームの値と一致しません。アプリコードで、ID トークンとアクセストークンを個別に検証します。署名を検証するまでは、アクセストークンのクレームを信用しないでください。詳細については、「[JSON ウェブトークンの検証](#)」を参照してください。アクセストークンの有効期限は、5 分から 1 日までの間で任意の値に設定できます。この値は、アプリケーションのクライアントごとに設定できます。

Important

マネージドログインを使用する場合、アクセストークンと ID トークンの最小の有効期間を 1 時間未満に設定しないでください。マネージドログインは、1 時間有効なブラウザ Cookie を設定します。アクセストークンの有効期間を 1 時間未満に設定しても、マネージドログイン Cookie の有効性には影響がなく、ユーザーは初回サインインから 1 時間にわたって追加の認証情報なしで再認証できます。

アクセストークンのヘッダー

ヘッダーには 2 つの情報 (キー ID: kid、アルゴリズム: alg) が含まれています。

```
{
```

```
"kid" : "1234example="
"alg" : "RS256",
}
```

kid

キー ID。その値は、トークンの JSON Web 署名 (JWS) をセキュア化するために使用されたキーを示します。ユーザープール署名キー ID は `jwtks_uri` エンドポイントで確認できます。

kid パラメータの詳細については、「[Key identifier \(kid\) header parameter](#)」を参照してください。

alg

Amazon Cognito が、アクセストークンをセキュア化するために使用した暗号化アルゴリズム。ユーザープールは、SHA-256 による RSA 署名である RS256 暗号化アルゴリズムを使用します。

alg パラメータの詳細については、「[Algorithm \(alg\) header parameter](#)」を参照してください。

アクセストークンのデフォルトペイロード

以下は、アクセストークンのサンプルペイロードです。詳細については、「[JWT claims](#)」を参照してください。[トークン生成前の Lambda トリガー](#) を使用して独自の設計のクレームを追加できます。

```
<header>.
{
  "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "device_key": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "cognito:groups": [
    "testgroup"
  ],
  "iss": "https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",
  "version": 2,
  "client_id": "xxxxxxxxxxxxexample",
  "aud": "https://api.example.com",
  "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "event_id": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "token_use": "access",
  "scope": "phone openid profile resourceserver.1/appclient2 email",
  "auth_time": 1676313851,
```

```
"exp":1676317451,
"iat":1676313851,
"jti":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
"username":"my-test-user"
}
.<token signature>
```

sub

認証されたユーザーの固有識別子 ([UUID](#)) またはサブジェクト。ユーザー名はユーザープール内で一意ではない可能性があります。sub クレームは、特定のユーザーを識別する最良の方法です。

cognito:groups

ユーザーをメンバーとするユーザープールグループの名前の配列。

iss

トークンを発行した ID プロバイダー。クレームは以下のような形式になります。

`https://cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE`

client_id

ユーザーを認証したユーザープールアプリケーション。Amazon Cognito は、ID トークン aud クレームで同じ値をレンダリングします。

aud

アクセストークンが認可する API の URL。アプリケーションが [リソースバインディング](#) を認可サーバーにリクエストした場合にのみ使用されます。

origin_jti

ユーザーの更新トークンに関連付けられたトークン失効識別子。Amazon Cognito は、[エンドポイントの取り消し](#) または [RevokeToken](#) API オペレーションを使用してユーザーのトークンが取り消されたかどうかを確認するときに、origin_jti クレームを参照します。トークンを取り消すと、Amazon Cognito は同じ origin_jti 値を持つすべてのアクセストークンと ID トークンを検証しなくなります。

token_use

トークンの本来の目的。アクセストークンでは、その値は access です。

scope

サインインしたユーザーに発行された OAuth 2.0 スコープのリスト。スコープは、userInfo エンドポイントにある外部 API、ユーザーによるセルフサービスのオペレーション、ユーザーデータに対する、トークンが提供するアクセス権を定義します。[トークンエンドポイント](#) のトークンには、アプリケーションがサポートする任意のスコープを含めることができます。Amazon Cognito API サインインからのトークンにはスコープ `aws.cognito.signin.user.admin` のみが含まれます。

auth_time

ユーザーが認証を完了した認証時刻 (Unix の時間形式)。

exp

ユーザーのトークンの有効期限が切れる有効期限 (Unix の時間形式)。

iat

Amazon Cognito がユーザーのトークンを発行した発行時刻 (Unix の時間形式)。

jti

JWT の一意の識別子。

username

ユーザープール内のユーザーのユーザー名。

その他のリソース

- [Amazon Cognito ユーザープールでアクセストークンをカスタマイズする方法](#)

アクセストークンの署名

`.well-known/jwks.json` エンドポイントでアドバタイズされたキーで署名されたアクセストークンの署名は、トークンヘッダーとペイロードの整合性を検証します。アクセストークンを使用して外部 API へのアクセスを許可する場合は、この署名を署名元のキーと照合して検証するように API オーソライザーを必ず設定します。詳細については、「[JSON ウェブトークンの検証](#)」を参照してください。

更新トークン

更新トークンを使用して、新しい ID およびアクセストークンを取得できます。更新トークンはデフォルトで、アプリケーションユーザーがユーザープールにサインインしてから 30 日後に有効期限が切れます。ユーザープールのアプリケーションを作成するときは、アプリケーションの更新トークンの有効期限を 60 分から 10 年までの任意の値に設定できます。

更新トークンによる新しいアクセストークンと ID トークンの取得

マネージドログインの認証コードフローを使用した認証や、API オペレーションまたは SDK メソッドを使用した認証が成功すると、Amazon Cognito は更新トークンを発行します。更新トークンは、新しい ID トークンとアクセストークン、さらに必要に応じて新しい更新トークンを返します。更新トークンは、以下の方法で使用できます。

GetTokensFromRefreshToken

[GetTokensFromRefreshToken](#) API オペレーションは、有効な更新トークンから新しい ID トークンとアクセストークンを発行します。更新トークンのローテーションを有効にしている場合は、新しい更新トークンも取得できます。

InitiateAuth と AdminInitiateAuth

[AdminInitiateAuth](#) または [InitiateAuth](#) API オペレーションには、REFRESH_TOKEN_AUTH 認証フローが含まれています。このフローでは、更新トークンを渡して新しい ID トークンとアクセストークンを取得します。[更新トークンのローテーション](#)が有効になっているアプリケーションクライアントでは、REFRESH_TOKEN_AUTH で認証することはできません。

OAuth トークンエンドポイント

[ドメイン](#)を持つユーザープールの[トークンエンドポイント](#)には、refresh_token 付与タイプがあります。これを通じて、有効な更新トークンから新しい ID トークン、アクセストークン、さらに必要に応じて ([更新トークンのローテーション](#)により) 更新トークンを発行します。

更新トークンのローテーション

必要に応じて、アプリケーションクライアントで更新トークンのローテーションを設定できます。更新トークンのローテーションを使用すると、クライアントは元の更新トークンを無効にし、トークンが更新されるたびに新しい更新トークンを発行できます。この設定を有効にすると、すべての形式のトークン更新で成功したリクエストごとに、新しい ID トークン、アクセストークン、および更新トークンが返されます。この設定を無効にすると、トークン更新リクエストは新しいアクセストークン

ンと ID トークンのみを返し、元の更新トークンは有効なままになります。新しい更新トークンは、元の更新トークンの残りの期間有効になります。更新トークンをローテーションするか、元の更新トークンを引き継ぐように、[アプリケーションクライアント](#)を設定できます。短時間の再試行を許可するには、元の更新トークンの猶予期間を最大 60 秒に設定することもできます。

更新トークンのローテーションについて知っておくべきこと

- 更新トークンのローテーションを有効にすると、ユーザープールの JSON ウェブトークンに新しいクレームが追加されます。アクセストークンと ID トークンに `origin_jti` と `jti` クレームが追加されます。これらのクレームに伴って JWT のサイズが増加します。
- 更新トークンのローテーションは、認証フロー `REFRESH_TOKEN_AUTH` と互換性がありません。更新トークンのローテーションを実装するには、アプリケーションクライアントでこの認証フローを無効にし、[GetTokensFromRefreshToken](#) API オペレーションまたは同等の SDK メソッドを使用してトークン更新リクエストを送信するようにアプリケーションを設計する必要があります。
- 更新トークンのローテーションを非アクティブにすると、`GetTokensFromRefreshToken` または `REFRESH_TOKEN_AUTH` を使用してトークン更新リクエストを完了できます。
- ユーザープールで[デバイスの記憶](#)がアクティブな場合は、`GetTokensFromRefreshToken` リクエストでデバイスキーを指定する必要があります。アプリケーションが最初の認証リクエストで送信した確認済みデバイスキーがユーザーにない場合、Amazon Cognito は新しいデバイスキーを発行します。この設定でトークンを更新するには、デバイスキーを指定する必要があります (このデバイスキーは、`AuthParameters` で指定したものが、認証レスポンスで受け取ったものかは問いません)。
- `GetTokensFromRefreshToken` リクエストで、トークン生成前の Lambda トリガーを `ClientMetadata` に渡すことができます。このデータは、トリガーの入カイベントに渡され、Lambda 関数のカスタムロジックで利用できる追加のコンテキストを提供します。

セキュリティのベストプラクティスとして、アプリケーションクライアントで更新トークンのローテーションを有効にします。

Enable refresh token rotation (console)

次の手順では、アプリケーションクライアントで更新トークンのローテーションをオンまたはオフにします。この手順では、作成済みのアプリケーションクライアントが必要です。アプリケーションクライアントの作成方法の詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

更新トークンのローテーションを有効にするには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [アプリケーションクライアント] メニューに移動し、既存のアプリケーションクライアントを選択します。
5. ページの [アプリケーションクライアントに関する情報] セクションで [編集] を選択します。
6. [高度なセキュリティ設定] で、[更新トークンのローテーションを有効にする] オプションを見つけます。
7. ローテーションを有効にするには、チェックボックスをオンにします。ローテーションを無効にするには、チェックボックスをオフにします。
8. [更新トークンのローテーションの猶予期間] に、ローテーション後に元の更新トークンを取り消すまでの遅延時間として最大 60 秒を入力します。

Enable refresh token rotation (API)

更新トークンのローテーションを [CreateUserPoolClient](#) または [UpdateUserPoolClient](#) API リクエストで設定します。次のリクエスト本文の部分では、更新トークンのローテーションを有効にし、猶予期間を 10 秒に設定します。

```
"RefreshTokenRotation" : {  
  "Feature" : "ENABLED",  
  "RetryGracePeriodSeconds" : 10  
}
```

API および SDK トークンの更新

更新トークンを使用してユーザープール API で新しい ID トークンとアクセストークンを取得するには 2 つの方法があり、更新トークンのローテーションが有効になっているかどうかに応じて使い分けます。アプリケーションクライアントで更新トークンのローテーションが有効になっている場合は、[GetTokensFromRefreshToken](#) API オペレーションを使用します。アプリケーションクライアントで更新トークンのローテーションが有効になっていない場合は、[AdminInitiateAuth](#) API オペレーションまたは [InitiateAuth](#) API オペレーションの REFRESH_TOKEN_AUTH フローを使用します。

Note

ユーザーは、[マネージドログイン](#)または AWS SDKs および Amazon Cognito API オペレーションで構築したカスタムアプリケーションで、ユーザープールで認証できます。REFRESH_TOKEN_AUTH フローと GetTokensFromRefreshToken は、どちらもマネージドログインユーザーのトークン更新を完了できます。カスタムアプリケーションでのトークン更新は、マネージドログインセッションには影響しません。これらのセッションはブラウザ Cookie で設定され、1 時間有効です。GetTokensFromRefreshToken レスポンスは、新しい ID トークン、アクセストークン、必要に応じて更新トークンを発行しますが、マネージドログインのセッション Cookie は更新しません。REFRESH_TOKEN_AUTH は、更新トークンのローテーションが有効になっているアプリケーションクライアントでは使用できません。

GetTokensFromRefreshToken

[GetTokensFromRefreshToken](#) は、更新トークンで認可したリクエストから新しい ID トークン、アクセストークン、更新トークンを返します。次に示すのは、GetTokensFromRefreshToken のリクエスト本文の例です。このオペレーションへのリクエストにより、クライアントメタデータを Lambda トリガーに送信できます。

```
{
  "RefreshToken": "eyJjd123abcEXAMPLE",
  "ClientId": "1example23456789",
  "ClientSecret": "myappclientsecret123abc",
  "ClientMetadata": {
    "MyMetadataKey" : "MyMetadataValue"
  },
}
```

AdminInitiateAuth/InitiateAuth

更新トークンのローテーションが無効になっているときに更新トークンを使用するには、[AdminInitiateAuth](#) API オペレーションまたは [InitiateAuth](#) API オペレーションを使用します。AuthFlow パラメータの REFRESH_TOKEN_AUTH を渡します。AuthFlow の AuthParameters プロパティで、ユーザーの更新トークンを "REFRESH_TOKEN" の値として渡します。Amazon Cognito は、API リクエストがすべてのチャレンジを通過した後、新しい ID とアクセストークンを返します。

次に示すのは、InitiateAuth API または AdminInitiateAuth API を使用したトークン更新のリクエスト本文の例です。

```
{
  "AuthFlow": "REFRESH_TOKEN_AUTH",
  "ClientId": "1example23456789",
  "UserPoolId": "us-west-2_EXAMPLE",
  "AuthParameters": {
    "REFRESH_TOKEN": "eyJjd123abcEXAMPLE",
    "SECRET_HASH": "kT5acwCVrbD6JexhW3EQwnRSe6fLuPTRkEQ50athqv8="
  }
}
```

OAuth トークンの更新

更新トークンは、ドメインを設定したユーザープール内の [トークンエンドポイント](#) に送信することもできます。リクエスト本文には、refresh_token の grant_type 値とユーザーの更新トークンの refresh_token 値を含めます。

トークンエンドポイントへのリクエストは、更新トークンのローテーションが有効になっているアプリケーションクライアントと無効になっているアプリケーションクライアントの両方で使用できます。更新トークンのローテーションが有効になっている場合、トークンエンドポイントは新しい更新トークンを返します。

次に示すのは、更新トークンを使用したリクエストの例です。

```
POST /oauth2/token HTTP/1.1
Host: auth.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic ZGpj0Th1M2ppZWRTaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw
Content-Length: **

client_id=1example23456789&grant_type=refresh_token&refresh_token=eyJjd123abcEXAMPLE
```

更新トークンの取り消し

ユーザーに属する更新トークンを取り消すことができます。トークンの取り消しの詳細については、「[トークン取り消しによるユーザーセッションの終了](#)」を参照してください。

Note

更新トークンを取り消すと、Amazon Cognito がそのトークンを使用して更新リクエストから発行したすべての ID トークンとアクセストークンが取り消されます。

現在サインインしているすべてのセッションからユーザーをサインアウトさせるには、[GlobalSignOut](#) API リクエストまたは [AdminUserGlobalSignOut](#) API リクエストを使用してすべてのトークンを取り消します。ユーザーがサインアウトすると、以下の結果になります。

- ユーザーの新しいトークンの取得にユーザーの更新トークンを使用できない。
- ユーザーのアクセストークンは、トークン認証された API リクエストを行うことができない。
- 新しいトークンを取得するためにユーザーが再認証される必要がある。マネージドログインのセッション Cookie は自動的に期限切れにならないため、ユーザーは認証情報の入力を求められることなく、セッション Cookie を使用して再認証できます。マネージドログインのユーザーをサインアウトさせたら、ユーザーを[ログアウトエンドポイント](#)にリダイレクトします。ここで Amazon Cognito はユーザーのセッション Cookie を消去します。

更新トークンを使用すると、ユーザーのセッションをアプリケーション内で長期間維持できます。時間が経つと、ユーザーは更新トークンでサインインしたままになっているアプリケーションの承認を解除したくなるかもしれません。1つのセッションからユーザーをサインアウトさせるには、ユーザーの更新トークンを取り消します。ユーザーがすべての認証済みセッションからログアウトしたい場合は、[GlobalSignOut](#) API リクエストを生成します。アプリケーションは、[すべてのデバイスからサインアウト]などのオプションをユーザーに提供できます。GlobalSignOut は、ユーザーの有効な (変更されていない、有効期限が切れていない、取り消されていない) アクセストークンを受け入れます。この API はトークン認証されているため、あるユーザーがそれを使用して別のユーザーのサインアウトを開始することはできません。

ただし、認証情報を使用して承認する [AdminUserGlobalSignOut](#) API リクエストを生成 AWS して、すべてのデバイスから任意のユーザーをサインアウトできます。管理者アプリケーションは AWS、開発者認証情報を使用してこの API オペレーションを呼び出し、ユーザープール ID とユーザーのユーザー名をパラメータとして渡す必要があります。AdminUserGlobalSignOut API は、ユーザープール内の任意のユーザーをサインアウトできます。

AWS 認証情報またはユーザーのアクセストークンのいずれかを使用して認可できるリクエストの詳細については、「[認可モデル別にグループ化された API オペレーションのリスト](#)」を参照してください。

トークン取り消しによるユーザーセッションの終了

更新トークンとエンドユーザーセッションは、以下の方法で取り消すことができます。更新トークンを取り消すと、その更新トークンによってそれまでに発行されたすべてのアクセストークンが無効になります。ユーザーに発行されたその他の更新トークンは影響を受けません。

RevokeToken オペレーション

[RevokeToken](#) は、特定の更新トークンのすべてのアクセストークン (インタラクティブサインインの初期アクセストークンを含む) を取り消します。このオペレーションは、ユーザーの他の更新トークン、または他の更新トークンの子である ID トークンやアクセストークンには影響しません。

取り消しエンドポイント

[取り消しエンドポイント](#) は、指定した更新トークンと、この更新トークンで生成したすべての ID トークンとアクセストークンを取り消します。このエンドポイントは、インタラクティブサインインの初期アクセストークンも取り消します。このエンドポイントへのリクエストは、ユーザーの他の更新トークン、または他の更新トークンの子である ID トークンおよびアクセストークンには影響しません。

GlobalSignOut オペレーション

[GlobalSignOut](#) は、ユーザーがアクセストークンを使用して承認するセルフサービスオペレーションです。このオペレーションは、リクエスト元のユーザーの更新トークン、ID トークン、アクセストークンをすべて取り消します。

AdminUserGlobalSignOut オペレーション

[AdminUserGlobalSignOut](#) は、管理者が IAM 認証情報を使用して承認するサーバー側のオペレーションです。このオペレーションは、ターゲットユーザーの更新トークン、ID トークン、アクセストークンをすべて取り消します。

トークンの取り消しについて知っておくべきこと

- 更新トークンを取り消すリクエストには、そのトークンを取得するために使用したクライアント ID を含める必要があります。
- ユーザープールの JWT は、自己完結型であり、トークンの作成時に割り当てた署名と有効期限があります。取り消されたトークンは、トークンを必要とする Amazon Cognito API コールでは使用できません。ただし、JWT ライブラリを使用してトークンの署名と有効期限が検証された場合は、引き続き使用が可能です。

- トークンの取り消しは、新しいユーザープールクライアントの作成時にデフォルトで有効になります。
- 更新トークンは、トークンの取り消しが無効になっているアプリケーションクライアントでのみ取り消すことができます。
- トークンの取り消しを有効にした後は、Amazon Cognito JSON ウェブトークンに新しいクレームが追加され、アクセストークンと ID トークンに `origin_jti` と `jti` クレームが追加されます。これらのクレームにより、アプリケーションクライアントのアクセストークンと ID トークンのサイズが大きくなります。
- 以前にトークンの取り消しが無効になっていたアプリケーションクライアントでトークンの取り消しを無効にすると、取り消したトークンは再び有効になりません。
- [ユーザーアカウントを無効にする](#)と (更新トークンとアクセストークンが取り消され)、ユーザーアカウントを再度有効にしても、取り消したトークンは有効になりません。
- AWS マネジメントコンソール、AWS CLI または API を使用して新しいユーザープールクライアントを作成すると AWS、トークンの失効がデフォルトで有効になります。

トークンの取り消しを有効にする

既存のユーザープールクライアントのトークンを取り消す前に、トークンの取り消しを有効にする必要があります。または AWS API を使用して AWS CLI、既存のユーザープールクライアントのトークン失効を有効にできます。これを行うには、`aws cognito-idp describe-user-pool-client` CLI コマンドまたは `DescribeUserPoolClient` API 操作を呼び出して、アプリケーションクライアントから現在の設定を取得します。次に、または `UpdateUserPoolClient` CLI コマンド `aws cognito-idp update-user-pool-client` API オペレーションを呼び出します。アプリケーションクライアントの現在の設定を含め、`EnableTokenRevocation` パラメータを `true` に設定します。

Amazon Cognito API または AWS SDK でトークン失効を有効にしたアプリケーションクライアントを作成または変更するには、[CreateUserPoolClient](#) または [UpdateUserPoolClient](#) API リクエストに次のパラメータを含めます。

```
"EnableTokenRevocation": true
```

Amazon Cognito コンソールでトークンの取り消しを設定するには、ユーザープールの [アプリケーションクライアント] メニューでアプリケーションクライアントを選択します。[アプリケーションクライアントに関する情報] で [編集] ボタンを選択し、[高度な認証設定] でトークンの取り消しを有効または無効にします。

トークンを取り消す

更新トークンは、[RevokeToken](#) API リクエストを `aws cognito-idp revoke-token` CLI コマンドなどを使用して取り消すことができます。[エンドポイントの取り消し](#) を使用してトークンを取り消すこともできます。このエンドポイントは、ユーザープールにドメインを追加した後で利用可能になります。取り消しエンドポイントは、Amazon Cognito がホストするドメイン、あるいは独自のカスタムドメインの両方で使用できます。

次は、RevokeToken API リクエストの例の本文です。

```
{
  "ClientId": "1example23456789",
  "ClientSecret": "abcdef123456789ghijklexample",
  "Token": "eyJjdHkiOiJKV1QiEXAMPLE"
}
```

次は、カスタムドメインのユーザープールの `/oauth2/revoke` エンドポイントに対する cURL リクエストの例です。

```
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'
```

RevokeToken オペレーションと `/oauth2/revoke` エンドポイントには、アプリケーションクライアントにクライアントシークレットがある場合を除き、追加の認可は必要ありません。

JSON ウェブトークンの検証

JSON ウェブトークン (JWT) は、簡単にデコード、読み取り、変更ができます。アクセストークンが変更されると、権限エスカレーションのリスクが発生します。ID トークンを変更すると、偽装リスクが発生します。アプリケーションはユーザープールをトークン発行者として信頼していますが、ユーザーが転送中のトークンを妨害した場合はどうなりますか？アプリケーションが Amazon Cognito が発行したトークンと同じトークンを受信していることを確認する必要があります。

Amazon Cognito は、OpenID Connect (OIDC) 仕様の整合性と機密性の一部を使用するトークンを発行します。ユーザープールトークンは、有効期限、発行者、デジタル署名などのオブジェクトの有効性を示します。区切りの JWT の 3 番目で最後のセグメントである署名は、トークン検証の主要な

コンポーネントです。悪意のあるユーザーがトークンを変更することがありますが、アプリケーションがパブリックキーを取得して署名を比較すれば、トークンは一致しなくなります。OIDC 認証から JWT を処理するアプリケーションは、サインインごとにこの検証オペレーションを実行する必要があります。

このページでは、一般的かつ特定の JWT の認証を推奨します。アプリケーション開発は、さまざまなプログラミング言語とプラットフォームに及びます。Amazon Cognito は OIDC をパブリック仕様に十分近い方法で実装するため、任意のデベロッパー環境の信頼できる JWT ライブラリで検証要件を処理できます。

これらのステップでは、ユーザープール JSON Web トークン (JWT) の検証を説明します。

トピック

- [前提条件](#)
- [aws-jwt-verify によるトークンの検証](#)
- [トークンの理解と検査](#)

前提条件

このセクションのタスクは、ライブラリ、SDK、またはソフトウェアフレームワークで既に処理されている可能性があります。AWS SDKsは、アプリで Amazon Cognito ユーザープールトークンの処理と管理のためのツールを提供します。AWS Amplify には、Amazon Cognito トークンを取得および更新する関数が含まれています。

詳細については、次のページを参照してください。

- [Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#)
- [SDK を使用した Amazon Cognito ID プロバイダーのコード例 AWS SDKs](#)
- Amplify Dev Center の [高度なワークフロー](#)

JSON ウェブトークン (JWT) のデコードと検証用として、多数のライブラリが用意されています。サーバー側の API 処理用にトークンを手動で処理する場合、または他のプログラミング言語を使用している場合は、これらのライブラリが役に立ちます。「[OpenID foundation の JWT トークンでの作業のためのライブラリのリスト](#)」を参照してください。

aws-jwt-verify によるトークンの検証

Node.js アプリでは、は、ユーザーがアプリに渡すトークンのパラメータを検証するために[aws-jwt-verify ライブラリ](#) AWS を推奨します。aws-jwt-verify を使用すると、1 つ以上のユーザープールについて検証したいクレーム値を CognitoJwtVerifier に入力できます。確認できる値には次のものがあります。

- アクセストークンや ID トークンの形式が不正ではなく、期限切れでもなく、有効な署名が付いているもの。
- アクセストークンが、[正しいユーザープールとアプリクライアント](#)から取得されたもの。
- アクセストークンクレームに、[正しい OAuth 2.0 スコープ](#)が含まれているもの。
- アクセストークンと ID トークンに署名したキーが、[ユーザープールの JWKS URI の kid 署名キーと一致していること](#)。

JWKS URI には、ユーザーのトークンに署名した秘密鍵に関する公開情報が含まれています。ユーザープールの JWKS URI は、`https://cognito-idp.<Region>.amazonaws.com/<userPoolId>/.well-known/jwks.json` で確認できます。

Node.js アプリや AWS Lambda オーソライザーで使える詳細とサンプルコードについては、GitHub の [aws-jwt-verify](#) を参照してください。

トークンの理解と検査

トークン検査をアプリに統合する前に、Amazon Cognito が JWT を組み立てる方法を検討してください。ユーザープールからサンプルトークンを取得します。それらをデコードして詳細に調べて特性を理解し、何をいつ検証するかを決定します。例えば、あるシナリオではグループメンバーシップを検証し、別のシナリオではスコープを調べたい可能性があります。

以下のセクションでは、アプリを準備するときに Amazon Cognito JWT を手動で検査するプロセスについて説明します。

JWT の構造を確認します

JSON ウェブトークン (JWT) は、間に . (ドット) 区切り文字がある 3 つのセクションで構成されます。

ヘッダー

Amazon Cognito がトークンの署名に使用したキー ID、kid、および RSA アルゴリズム、alg。Amazon Cognito は RS256 の alg でトークン署名します。kid は、ユーザープールが保持する 2048 ビット RSA プライベート署名キーへの切り捨てられたリファレンスです。

ペイロード

トークンクレーム。ID トークンでは、クレームには、ユーザー属性とユーザープール、iss、およびアプリケーション、aud に関する情報が含まれます。アクセストークンのペイロードには、スコープ、グループメンバーシップ、ユーザープールが iss として含まれ、アプリケーションは client_id として含まれます。

Signature

署名は、ヘッダーやペイロードのように base64url でデコードできません。JWKS URI で確認できる署名キーとパラメータから派生した RSA256 識別子です。

ヘッダーとペイロードは base64url でエンコードされた JSON です。開始文字 { にデコードされる最初の文字 eyJ で識別できます。ユーザーが base64url でエンコードした JWT をアプリケーションに提示した形式が [JSON Header].[JSON Payload].[Signature] になっていない場合、これは有効な Amazon Cognito トークンではないため、破棄できます。

次のサンプルアプリケーションでは、aws-jwt-verify を使用してユーザープールのトークンを検証します。

```
// cognito-verify.js
// Usage example: node cognito-verify.js eyJra789ghiEXAMPLE

const { CognitoJwtVerifier } = require('aws-jwt-verify');

// Replace with your Amazon Cognito user pool ID
const userPoolId = 'us-west-2_EXAMPLE';

async function verifyJWT(token) {
  try {
    const verifier = CognitoJwtVerifier.create({
      userPoolId,
      tokenUse: 'access', // or 'id' for ID tokens
      clientId: '1example23456789', // Optional, only if you need to verify the token audience
    });
```

```
    const payload = await verifier.verify(token);
    console.log('Decoded JWT:', payload);
  } catch (err) {
    console.error('Error verifying JWT:', err);
  }
}

// Example usage
if (process.argv.length < 3) {
  console.error('Please provide a JWT token as an argument.');
```

```
  process.exit(1);
}

const MyToken = process.argv[2];
verifyJWT(MyToken);
```

JWT を検証

JWT 署名は、ヘッダーとペイロードのハッシュされた組み合わせです。Amazon Cognito は、ユーザープールごとに 2 組の RSA 暗号化キーを生成します。1 つの秘密鍵がアクセストークンに署名し、もう 1 つが ID トークンに署名します。

JWT トークンの署名を検証する

1. ID トークンを復号化します。

OpenID Foundation では、[JWT トークンでの作業のためのライブラリのリストも維持されています](#)。

AWS Lambda を使用してユーザープール JWTs デコードすることもできます。詳細については、「[「を使用した Amazon Cognito JWT トークンのデコードと検証 AWS Lambda」](#)」を参照してください。

2. ローカルキー ID (kid) とパブリック kid を比較します。

- a. ユーザープール用に、対応するパブリック JSON Web キー (JWK) をダウンロードして保存します。これは、JSON Web キーセット (JWKS) の一部として提供されており、環境に合わせて次のように `jwks_uri` URL を構築することで、その場所を特定できます。

```
https://cognito-idp.<Region>.amazonaws.com/<userPoolId>/.well-known/jwks.json
```

JWK と JWK セットの詳細については、「[JSON Web Key \(JWK\)](#)」を参照してください。

 Note

Amazon Cognito は、ユーザープール内で署名キーをローテーションする可能性があります。ベストプラクティスとして、kid をキャッシュキーとして使用して、アプリに公開鍵をキャッシュし、定期的にキャッシュを更新してください。アプリが受け取るトークンの kid をキャッシュと比較します。

正しい発行者で kid が異なるトークンを受け取った場合、Amazon Cognito が署名キーをローテーションした可能性があります。ユーザープール jwks_uri エンドポイントのキャッシュを更新します。

以下は、サンプル jwks.json ファイルです。

```
{
  "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**)

kid は、トークンの JSON ウェブ署名 (JWS) をセキュア化するために使用されたキーを示すヒントです。

Algorithm (**alg**)

alg ヘッダーパラメータは、ID トークンをセキュア化するために使用される暗号化アルゴリズムを表します。ユーザープールは、SHA-256 による RSA 署名である RS256 暗号化アルゴリズムを使用します。RSA の詳細については、「[RSA cryptography](#)」を参照してください。

Key type (**kt**)

kt パラメータは、この例の「RSA」など、キーで使用される暗号化アルゴリズムファミリーを特定します。

RSA exponent (**e**)

e パラメータには、RSA パブリックキーの指数値が含まれます。これは、Base64urlUInt でエンコードされた値として表されます。

RSA modulus (**n**)

n パラメータには、RSA パブリックキーのモジュラス値が含まれます。これは、Base64urlUInt でエンコードされた値として表されます。

Use (**use**)

use パラメータは、パブリックキーの用途を表します。この例では、use 値の sig が署名を表しています。

- b. JWT の kid に一致する kid のパブリック JSON ウェブキーを検索します。

クレームを検証する

JWT クレームを検証する

1. 次のいずれかの方法で、トークンの有効期限が切れていないことを確認します。
 - a. トークンをデコードし、exp クレームを現在の時刻と比較します。
 - b. アクセストークンに `aws.cognito.signin.user.admin` クレームが含まれている場合は、[GetUser](#) などの API にリクエストを送信します。[アクセストークンで承認](#)する API リクエストは、トークンの有効期限が切れているとエラーを返します。
 - c. [userInfo エンドポイント](#) へのリクエストでアクセストークンを提示します。トークンの有効期限が切れている場合、リクエストはエラーを返します。

2. ID トークンの `aud` クレームとアクセストークンの `client_id` クレームは、Amazon Cognito ユーザープールで作成されたアプリケーション ID と一致する必要があります。
3. 発行者 (`iss`) のクレームは、ユーザープールと一致する必要があります。例えば、`us-east-1` リージョンで作成されたユーザープールには、以下の `iss` 値があります。

`https://cognito-idp.us-east-1.amazonaws.com/<userpoolID>`.

4. `token_use` クレームをチェックします。
 - Web API オペレーションでアクセストークンのみを受け入れている場合は、その値を `access` にする必要があります。
 - ID トークンのみを使用している場合、その値は `id` にする必要があります。
 - ID とアクセストークンのいずれも使用している場合、`token_use` クレームは、`id` または `access` になります。

これで、トークン内のクレームを信頼できるようになりました。

ユーザープールトークンの有効期限とキャッシュの管理

新しい JSON ウェブトークン (JWT) を取得するたびに、アプリは次のいずれかのリクエストを正常に完了する必要があります。

- [トークンエンドポイント](#) からクライアント認証情報または認可コードの[付与](#)をリクエストする。
- マネージドログインページに対して暗黙的な付与をリクエストします。
- [InitiateAuth](#) のような Amazon Cognito API リクエストでローカルユーザーを認証する。

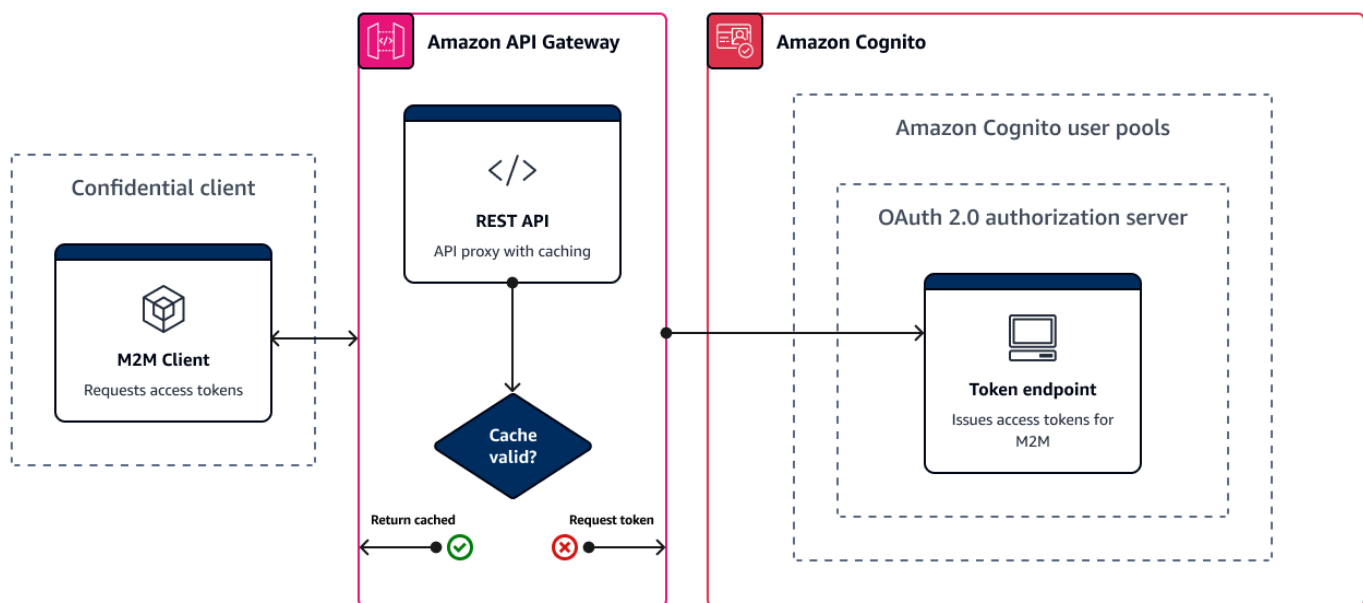
トークンの有効期限が分単位、時間単位、または日単位になるようにユーザープールを設定できます。アプリケーションのパフォーマンスと可用性を確保するには、トークンの有効期間の約 75% まで Amazon Cognito トークンを使用してから、新しいトークンを取得してください。アプリ用に構築したキャッシュソリューションはトークンを利用可能な状態に保ち、リクエストレートが高すぎる場合に Amazon Cognito がリクエストを拒否するのを防ぎます。クライアント側のアプリは、トークンをメモリキャッシュに保存する必要があります。サーバー側アプリでは、暗号化されたキャッシュメカニズムを追加してトークンを保存できます。

ユーザープールがユーザーまたはマシンツーマシンのアクティビティを大量に生成する場合、Amazon Cognito が設定したトークンをリクエストできる数の制限に遭遇する可能性があります。Amazon Cognito エンドポイントへのリクエスト数を減らすには、認証データを安全に保存して再利用するか、エクスポンネンシャルバックオフと再試行を実装することができます。

認証データは 2 つのクラスのエンドポイントから取得されます。Amazon Cognito の [OAuth 2.0 エンドポイント](#) に含まれているトークンエンドポイントでは、クライアントの認証情報とマネージドログインの認証コードのリクエストを処理します。[サービスエンドポイント](#) は、InitiateAuth や RespondToAuthChallenge のようなユーザープール API リクエストに応答します。各タイプのリクエストには独自の制限があります。制限事項の詳細については、「[Amazon Cognito のクォータ](#)」を参照してください。

Amazon API Gateway によるマシンツーマシンアクセストークンのキャッシュ

API Gateway トークンキャッシュを使用すると、Amazon Cognito OAuth エンドポイントのデフォルトのリクエストレートクォータを超えるイベントに応じてアプリをスケールさせることができます。



アクセストークンをキャッシュして、キャッシュされたトークンの有効期限が切れた場合にのみ、アプリが新しいアクセストークンをリクエストするようにできます。それ以外の場合、キャッシュエンドポイントはキャッシュからトークンを返します。これにより、Amazon Cognito API エンドポイントへの追加呼び出しを防ぐことができます。Amazon API Gateway を [トークンエンドポイント](#) へのプロキシとして使用する場合、API はリクエストクォータに影響するリクエストの大半に応答し、レート制限によるリクエストの失敗を防ぎます。

次の API Gateway ベースのソリューションでは、トークンキャッシュの低レイテンシー、ローコード、ノーコード実装が提供されます。API Gateway API は転送中に暗号化され、オプションで保存時にも暗号化されます。API Gateway キャッシュは、OAuth 2.0 [クライアント認証情報の付与](#) に最適で、マシンツーマシンセッションとマイクロサービスセッションを認可するためのアクセストークンを生成し、頻繁かつ大量に付与します。マイクロサービスが水平方向にスケールするトラフィック

の急増などのイベントでは、ユーザープールまたはアプリケーションクライアントの AWS リクエストレート制限を超えるボリュームで同じクライアント認証情報を使用する多くのシステムが発生する可能性があります。このようなシナリオでは、アプリの可用性と低レイテンシーを維持するために、キャッシュソリューションがベストプラクティスです。

このソリューションでは、API にキャッシュを定義して、アプリでリクエストする OAuth スコープとアプリケーションクライアントの組み合わせごとに個別のアクセストークンを保存します。アプリがキャッシュキーと一致するリクエストを行うと、API はキャッシュキーと一致する最初のリクエストに対して Amazon Cognito が発行したアクセストークンを返します。キャッシュキーの有効期限が切れると、API はリクエストをトークンエンドポイントに転送し、新しいアクセストークンをキャッシュします。

 Note

キャッシュキーの有効期間は、アプリケーションクライアントのアクセストークン有効期間よりも短くする必要があります。

キャッシュキーは、リクエスト本文の scope URL パラメータでリクエストした OAuth スコープと、リクエストの Authorization ヘッダーの組み合わせです。Authorization ヘッダーには、アプリのクライアント ID とクライアントシークレットが含まれます。このソリューションを実装するために、アプリに追加のロジックを実装する必要はありません。ユーザープールトークンエンドポイントへのパスを変更するには、設定を更新するだけで済みます。

トークンキャッシュは、[ElastiCache \(Redis OSS\)](#) で実装することもできます。AWS Identity and Access Management (IAM) ポリシーによるきめ細かなコントロールが必要な場合は、[Amazon DynamoDB](#) キャッシュをご検討ください。

 Note

API Gateway でのキャッシュには追加料金がかかります。[詳細については、料金表を参照してください。](#)

API Gateway でキャッシュプロキシをセットアップする方法

1. [API Gateway コンソール](#)を開き、REST API を作成します。
2. [Resources] (リソース) で、POST メソッドを作成します。

- a. HTTP [Integration type] (統合タイプ) を選択してください。
 - b. [Use HTTP proxy integration] (HTTP プロキシ統合の使用) を選択します。
 - c. `https://<your user pool domain>/oauth2/token` の [Endpoint URL] (エンドポイント URL) を入力します。
3. [Resources] (リソース) で、キャッシュキーを設定します。
 - a. POST メソッドの [Method request] (メソッドリクエスト) を編集します。

 Note

このメソッドリクエストの検証は、トークンリクエストの `client_secret_basic` 認可で使われます。これに伴って、クライアントシークレットが Authorization リクエストのヘッダーにエンコードされます。`client_secret_post` 認可における JSON リクエスト本文の検証では、代わりに `client_secret` が存在することを必須とする [データモデル](#) を作成します。このモデルでは、[リクエストの検証] で [本文、クエリ文字列パラメータ、およびヘッダーの検証] を行う必要があります。

- b. [リクエストの検証] メソッドを [クエリ文字列パラメータおよびヘッダーの検証] に設定します。リクエストの検証の詳細については、「Amazon API Gateway デベロッパーガイド」の「[リクエストの検証](#)」を参照してください。
 - c. scope パラメータと Authorization ヘッダーをキャッシュキーとして設定します。
 - i. [URL クエリ文字列パラメータ] にクエリ文字列を追加します。scope の [名前] にクエリ文字列名を入力し、[必須] と [キャッシュ] を選択します。
 - ii. [HTTP リクエストヘッダー] にヘッダーを追加します。Authorization の [名前] にリクエストヘッダー名を入力し、[必須] と [キャッシュ] を選択します。
4. [Stages] (ステージ) で、キャッシュを設定します。
 - a. 変更するステージを選択し、[ステージの詳細] で [編集] を選択します。
 - b. [その他の設定] の [キャッシュ設定] で、[API キャッシュをプロビジョニング] オプションをオンにします。
 - c. [Cache capacity] (キャッシュ容量) を選択します。キャッシュキャパシティが大きいほどパフォーマンスは向上しますが、追加コストが発生します。

- d. [承認を必須にする] チェックボックスをオフにします。[Continue] (続行) をクリックします。
- e. API Gateway は、ステージレベルの GET メソッドにのみキャッシュポリシーを適用します。POST メソッドには、キャッシュポリシーのオーバーライドを適用する必要があります。

設定したステージを展開し、POST メソッドを選択します。メソッドのキャッシュ設定を作成するには、[オーバーライドを作成] を選択します。

- f. [メソッドキャッシュを有効にする] オプションを有効にします。
 - g. [キャッシュの有効期限 (TTL)] として 3,600 秒を入力します。[保存] を選択します。
5. [Stages] (ステージ) で、[Invoke URL] (URL を呼び出す) に注目します。
6. ユーザープールの /oauth2/token エンドポイントの代わりに、API の Invoke URL (URL を呼び出す) にトークンリクエストを POST するようにアプリを更新します。

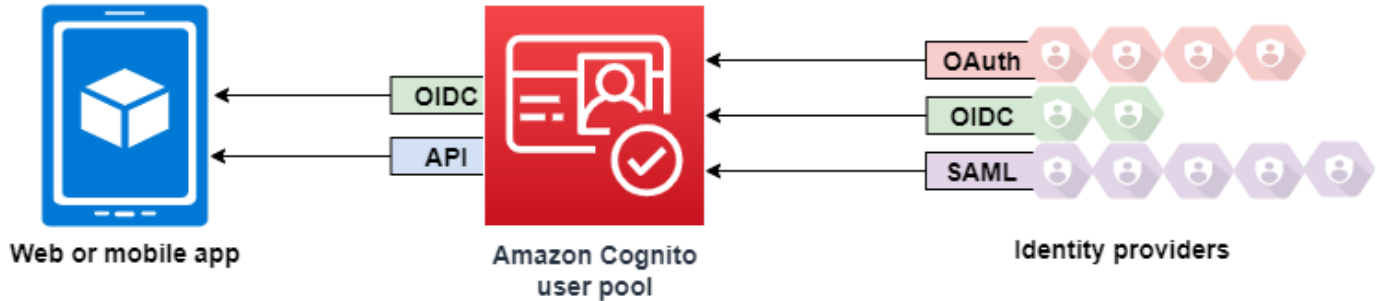
サインイン完了後にリソースにアクセスする

アプリケーションユーザーは、ユーザープール経由で直接サインインするか、またはサードパーティーの ID プロバイダー (IdP) 経由でフェデレーション方式で認証を行うことができます。ユーザープールは、Facebook、Google、Amazon、Apple 経由のソーシャルサインイン、および OpenID Connect (OIDC) と SAML IdP から返されるトークンの処理のオーバーヘッドを管理します。詳細については、「[ユーザープール JSON ウェブトークン \(JWT\) の理解](#)」を参照してください。

認証が正常に行われると、アプリケーションが Amazon Cognito からユーザープールトークンを受け取ります。ユーザープールトークンを使用すると、次のことができます。

- Amazon DynamoDB や Amazon S3 AWS のサービス などの のアプリケーションリソースのリクエストを承認する AWS 認証情報を取得します。
- 一時的で取り消し可能な認証の証拠を提供します。
- アプリケーションのユーザープロフィールに ID データを入力します。
- ユーザープールディレクトリでサインインしたユーザーのプロファイルの変更を認可します。
- アクセストークンを使用して、ユーザー情報のリクエストを認可します。
- アクセストークンを使用して、アクセス保護された外部 API の背後にあるデータへのリクエストを認可します。
- Amazon Verified Permissions を使用して、クライアントまたはサーバーに保存されているアプリケーションアセットへのアクセスを認可します。

詳細については、「[認証セッションの例](#)」および「[ユーザープール JSON ウェブトークン \(JWT\) の理解](#)」を参照してください。



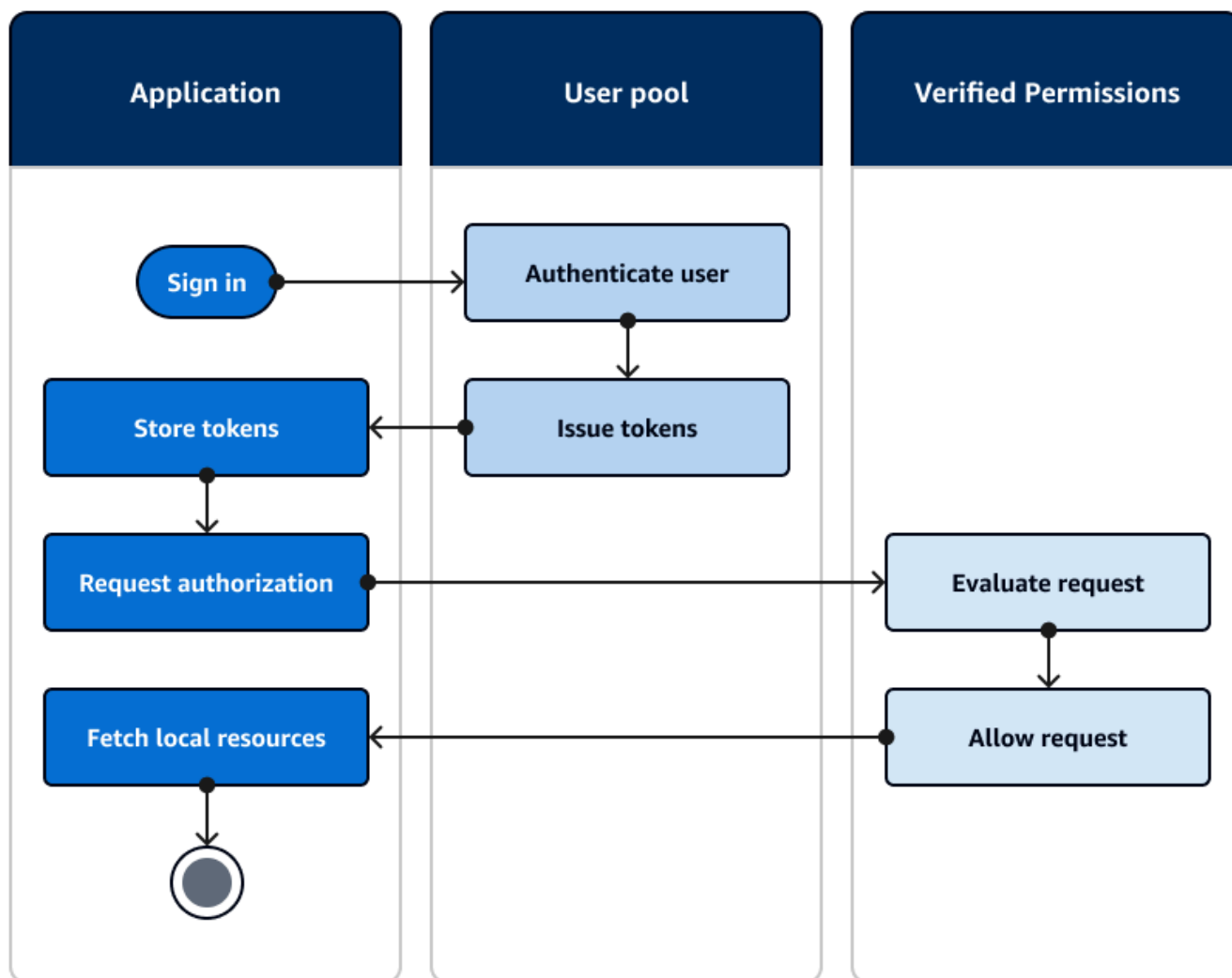
トピック

- [Amazon Verified Permissions を使用したクライアントまたはサーバーリソースへのアクセスの認可](#)
- [サインイン後の API Gateway を使用したリソースへのアクセス](#)
- [サインイン後の ID プール AWS のサービス を使用した へのアクセス](#)

Amazon Verified Permissions を使用したクライアントまたはサーバーリソースへのアクセスの認可

アプリケーションは、サインインしたユーザーから [Amazon Verified Permissions](#) にトークンを渡すことができます。Verified Permissions は、構築したアプリケーション向けの、スケーラブルできめ細かなアクセス許可管理および認可サービスです。Amazon Cognito ユーザープールは、Verified Permissions ポリシーストアの ID ソースにすることができます。Verified Permissions は、ユーザープールトークンのプリンシパルとその属性から、premium_badge.png、GetPhoto などのリクエストされたアクションとリソースの認可に関する決定を行います。

次の図は、アプリケーションが認可リクエストでユーザーのトークンを Verified Permissions に渡す方法を示しています。



Amazon Verified Permissions の使用開始

ユーザープールを Verified Permissions と統合すると、すべての Amazon Cognito アプリケーションに対する詳細な認のを一元的ソースを取得できます。これにより、すべてのアプリケーション間でコーディングとレプリケートを行う必要があるきめ細かなセキュリティロジックが不要になります。Verified Permissions による認可の詳細については、「[Amazon Verified Permissions による承認](#)」を参照してください。

Verified Permissions 認可リクエストには AWS 認証情報が必要です。認可リクエストに認証情報を安全に適用するには、以下の手法の一部を実装できます。

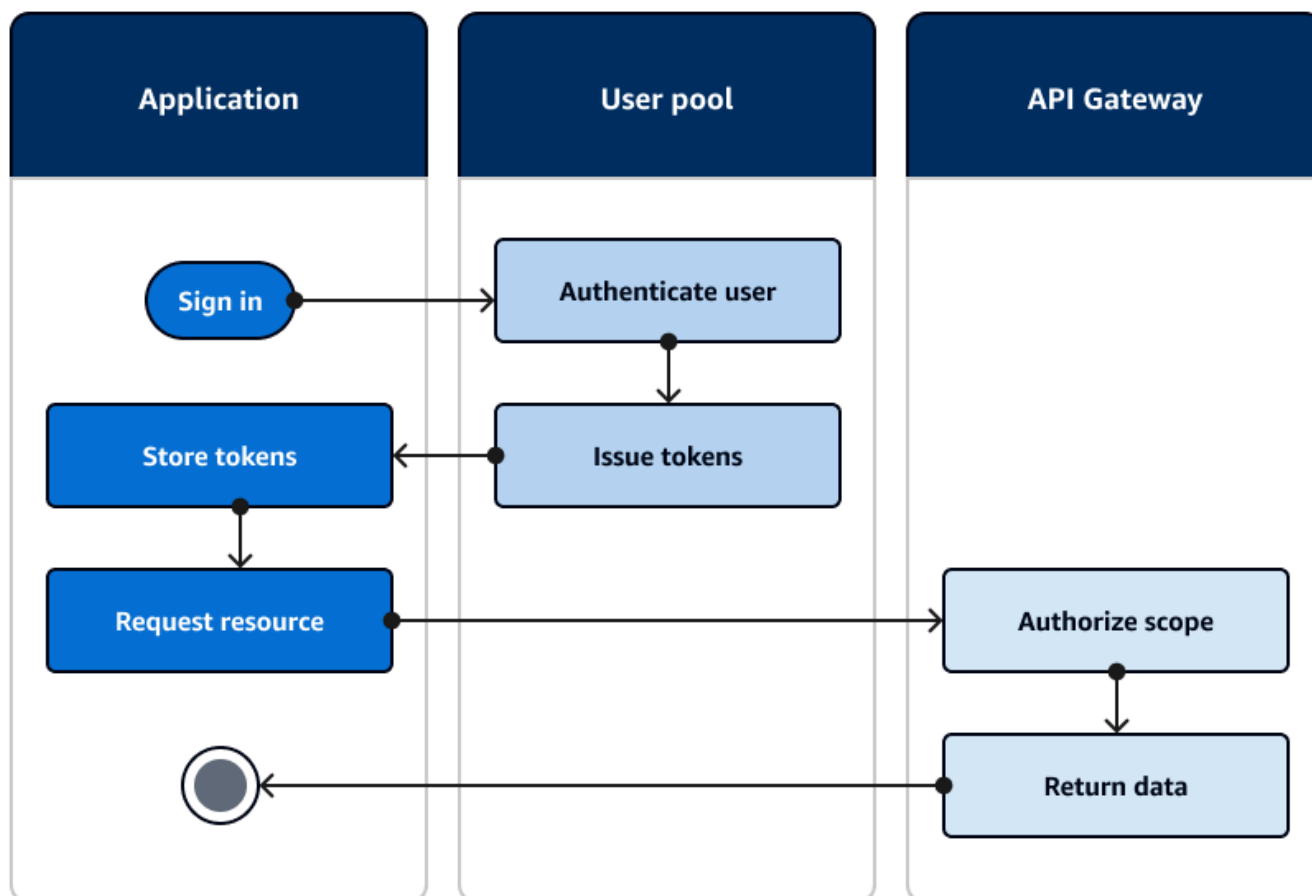
- サーバーのバックエンドにシークレットを保存できるウェブアプリケーションを運用します。

- 認証されたアイデンティティプールの認証情報を取得します。
- アクセストークン認証 API を使用してユーザーリクエストをプロキシし、リクエストに AWS 認証情報を追加します。

サインイン後の API Gateway を使用したリソースへのアクセス

Amazon Cognito ユーザープールトークンの一般的な用途は、[API Gateway REST API](#) へのリクエストを認可することです。アクセストークンの OAuth 2.0 スコープは、/app_assets 用の HTTP GET のようなメソッドとパスを認可できます。ID トークンは API への汎用認証として機能し、ユーザー属性をバックエンドサービスに渡すことができます。API Gateway には、[HTTP API 用の JWT オーソライザー](#)や [Lambda オーソライザー](#)などの追加のカスタム認証オプションがあり、よりきめ細かなロジックを適用できます。

次の図は、アクセストークンの OAuth 2.0 スコープを使用して REST API へのアクセスを取得しているアプリケーションを示しています。



アプリケーションは、認証されたセッションからトークンを収集し、それらをベアラトークンとしてリクエストの Authorization ヘッダーに追加する必要があります。API、パス、およびメソッド用に設定したオーソライザーを設定して、トークンの内容を評価します。API Gateway は、オーソライザー用に設定した条件にリクエストが一致する場合にのみデータを返します。

API Gateway API がアプリケーションからのアクセスを承認できる可能性のある方法は次のとおりです。

- アクセストークンは有効であり、有効期限が切れておらず、正しい OAuth 2.0 スコープが含まれています。[REST API 用の Amazon Cognito ユーザープールオーソライザー](#)は、エントリに対する障壁が低い一般的な実装です。また、このタイプのオーソライザーへのリクエストの本文、クエリ文字列パラメータ、およびヘッダーを評価することもできます。
- ID トークンは有効であり、有効期限が切れていません。ID トークンを Amazon Cognito オーソライザーに渡すと、アプリケーションサーバーで ID トークンの内容をさらに検証できます。
- アクセストークンまたは ID トークンのグループ、クレーム、属性、ロールは、Lambda 関数で定義した要件を満たします。[Lambda オーソライザー](#)は、リクエストヘッダー内のトークンを解析し、認可に関する決定のために評価します。関数にカスタムロジックを構築するか、[Amazon Verified Permissions](#) に API リクエストを行うことができます。

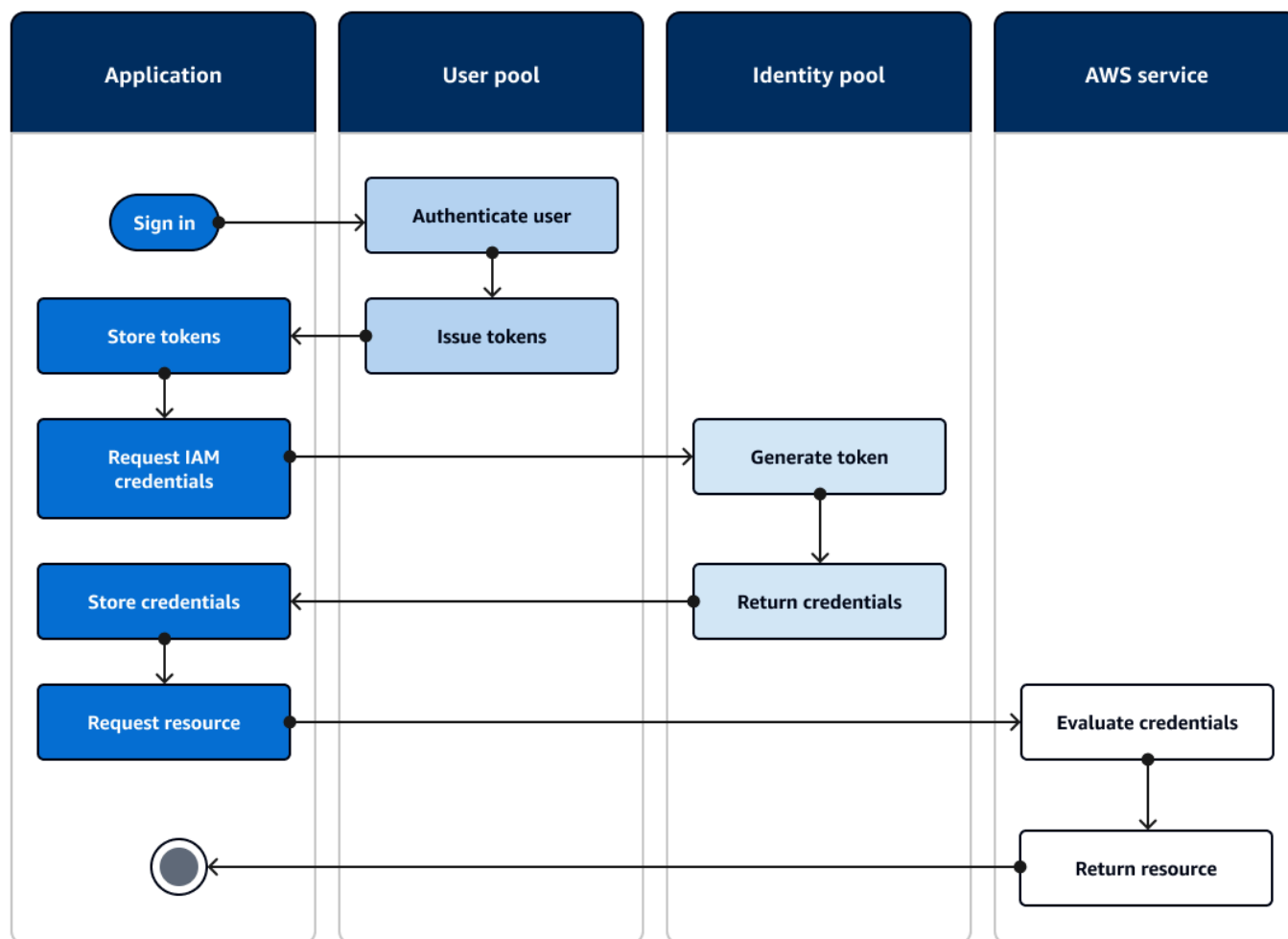
また、ユーザープールからのトークンを使用して [AWS AppSync GraphQL API](#) へのリクエストを認可することもできます。

サインイン後の ID プール AWS のサービス を使用した へのアクセス

ユーザーがユーザープールでサインインすると、ID プールから発行された一時的な API 認証情報 AWS のサービス を使用して にアクセスできます。

ウェブアプリケーションまたはモバイルアプリケーションは、ユーザープールからトークンを受け取ります。ユーザープールを ID プールの ID プロバイダーとして設定すると、ID プールはトークンを一時的な AWS 認証情報と交換します。これらの認証情報は、限られた AWS リソースセットへのアクセスをユーザーに付与する IAM ロールとそのポリシーに限定できます。詳細については、「[ID プールの認証フロー](#)」を参照してください。

次の図は、アプリケーションがユーザープールでサインインし、アイデンティティプールの認証情報を取得し、AWS のサービスにアセットをリクエストする方法を示しています。



アイデンティティプールの認証情報を使用すると、次のことができます。

- ユーザー自身の認証情報を使用して、Amazon Verified Permissions にきめ細かな認証リクエストを行います。
- IAM との接続を許可する Amazon API Gateway REST API または AWS AppSync GraphQL API に接続します。
- IAM への接続を認可する Amazon DynamoDB や Amazon RDS などのデータベースバックエンドに接続します。
- Amazon S3 バケットからアプリケーションアセットを取得します。
- Amazon WorkSpaces 仮想デスクトップでセッションを開始します。

アイデンティティプールは、ユーザープールを用いて認証されたセッション内でのみ動作するわけではありません。また、サードパーティー ID プロバイダーから直接認証を受け入れ、認証されていないゲストユーザーの認証情報を生成することもできます。

ID プールをユーザープールグループと一緒に使用して AWS リソースへのアクセスを制御する方法の詳細については、[ユーザープールにグループを追加する](#)「」および「」を参照してください。[ロールベースアクセスコントロールの使用](#)。また、ID プールの詳細については AWS Identity and Access Management、「」を参照してください。[ID プールの認証フロー](#)。

を使用したユーザープールの設定 AWS マネジメントコンソール

Amazon Cognito ユーザープールを作成し、クライアントアプリごとに [User Pool ID] (ユーザープール ID) と [App Client ID] (アプリクライアント ID) をメモしておきます。ユーザープール作成の詳細については、「[ユーザープールの開始方法](#)」を参照してください。

を使用した ID プールの設定 AWS マネジメントコンソール

次の手順では、を使用して ID プール AWS マネジメントコンソール を 1 つ以上のユーザープールおよびクライアントアプリケーションと統合する方法について説明します。

Amazon Cognito のユーザープール ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Amazon Cognito ユーザープール] を選択します。
5. ユーザープール ID とアプリクライアント ID を入力します。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - a. その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを与えることも、ルール付きのロールを選択することもできます。Amazon Cognito ユーザープール IdP では、トークンの優先ロールクレームを持つロールを選択することもできます。cognito:preferred_role クレームの詳細については、「[グループへの優先順位の値の割り当て](#)」を参照してください。
 - i. [ルールを使用してロールを選択する] を選択した場合、ユーザー認証からのソースクレーム、ルールへのクレームを比較するために使用するオペレータ、このロール選択と

一致する値、およびロール割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。

- ii. [トークンの優先ロールクレームを持つロールを選択] を選択した場合、Amazon Cognito はユーザーの `cognito:preferred_role` クレームでロールの認証情報を発行します。優先ロールクレームが存在しない場合、Amazon Cognito はロール解決に基づいて認証情報を発行します。

- b. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。

7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。

- プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
- `sub` および `aud` クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
- プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。

8. [変更を保存] を選択します。

ユーザープールを ID プールと統合する

アプリユーザーが認証されると、認証情報プロバイダーのログインマップにユーザーのアイデンティティトークンを追加します。プロバイダー名は、Amazon Cognito ユーザープール ID に応じて異なります。次の構造になります。

```
cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>
```

<region> の値は、ユーザープール ID から取得できます。例えば、ユーザープール ID が `us-east-1_EXAMPLE1` の場合、**<region>** は `us-east-1` になります。ユーザープール ID が `us-west-2_EXAMPLE2` の場合、**<region>** は `us-west-2` になります。

JavaScript

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

if (cognitoUser != null) {
```



```

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



```

    // Add the User's Id Token to the Cognito credentials login map.
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'YOUR_IDENTITY_POOL_ID',
      Logins: {
        'cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>':
result.getIdToken().getJwtToken()
      }
    });
  }
});
}

```

Android

```

cognitoUser.getSessionInBackground(new AuthenticationHandler() {
  @Override
  public void onSuccess(CognitoUserSession session) {
    String idToken = session.getIdToken().getJWTToken();

    Map<String, String> logins = new HashMap<String, String>();
    logins.put("cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>",
session.getIdToken().getJWTToken());
    credentialsProvider.setLogins(logins);
  }

});

```

iOS - objective-C

```

AWSServiceConfiguration *serviceConfiguration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSEast1 credentialsProvider:nil];
AWSCognitoIdentityUserPoolConfiguration *userPoolConfiguration =
[[AWSCognitoIdentityUserPoolConfiguration alloc] initWithClientId:@"YOUR_CLIENT_ID"
clientSecret:@"YOUR_CLIENT_SECRET" poolId:@"YOUR_USER_POOL_ID"];
[AWSCognitoIdentityUserPool
registerCognitoIdentityUserPoolWithConfiguration:serviceConfiguration
userPoolConfiguration:userPoolConfiguration forKey:@"UserPool"];
AWSCognitoIdentityUserPool *pool = [AWSCognitoIdentityUserPool
CognitoIdentityUserPoolForKey:@"UserPool"];

```

```
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider alloc] initWithRegionType:AWSRegionUSEast1 identityPoolId:@"YOUR_IDENTITY_POOL_ID" identityProviderManager:pool];
```

iOS - swift

```
let serviceConfiguration = AWSServiceConfiguration(region: .USEast1, credentialsProvider: nil)
let userPoolConfiguration = AWSCognitoIdentityUserPoolConfiguration(clientId: "YOUR_CLIENT_ID", clientSecret: "YOUR_CLIENT_SECRET", poolId: "YOUR_USER_POOL_ID")
AWSCognitoIdentityUserPool.registerCognitoIdentityUserPoolWithConfiguration(serviceConfiguration, userPoolConfiguration: userPoolConfiguration, forKey: "UserPool")
let pool = AWSCognitoIdentityUserPool(forKey: "UserPool")
let credentialsProvider = AWSCognitoCredentialsProvider(regionType: .USEast1, identityPoolId: "YOUR_IDENTITY_POOL_ID", identityProviderManager:pool)
```

スコープ、M2M、リソースサーバー

ユーザープールのドメインを設定すると、Amazon Cognito は OAuth 2.0 認可サーバーおよびホストされたウェブ UI を自動的にプロビジョニングします。このウェブ UI により、アプリケーションはユーザーにサインアップページとサインインページを表示できます。詳細については、「[ユーザープールのマネージドログイン](#)」を参照してください。認可サーバーによってアクセストークンに追加するスコープを選択できます。スコープはリソースサーバーとユーザーデータへのアクセスを許可します。

リソースサーバーとは、OAuth 2.0 API サーバーのことです。リソースサーバーは、アクセスが保護されたリソースを保護するために、保護対象の API でリクエストされたメソッドとパスを承認するスコープが、ユーザープールのアクセストークンに含まれていることを検証します。トークンの署名に基づく発行者の検証、トークンの有効期限に基づく有効性の検証、トークンクレームのスコープに基づくアクセスレベルの検証を行います。ユーザープールスコープは、アクセストークンの scope クレームにあります。Amazon Cognito アクセストークンのクレームの詳細については、「[アクセストークンの理解](#)」を参照してください。

Amazon Cognito を使用すると、アクセストークンのスコープは、外部 API またはユーザー属性へのアクセスを認可できます。アクセストークンは、ローカルユーザー、フェデレーションユーザー、またはマシン ID に発行できます。

トピック

- [API 認可](#)

- [Machine to machine \(M2M\) 認可](#)
- [スコープについて](#)
- [リソースサーバーについて](#)
- [リソースバインディング](#)

API 認可

以下は、Amazon Cognito トークンを使用して API へのリクエストを認可する方法の一部です。

アクセストークン

Amazon Cognito オーソライザーを REST API メソッドリクエスト設定に追加する場合は、オーソライザー設定に [認可スコープ] を追加します。この設定では、API は Authorization ヘッダー内のアクセストークンを受け入れたうえで、受け入れたスコープについてレビューします。

ID トークン

有効な ID トークンを REST API の Amazon Cognito オーソライザーに渡すと、API Gateway はリクエストを受け入れ、ID トークンの内容を API バックエンドに渡します。

Amazon Verified Permissions

Verified Permissions では、[API にリンクされたポリシーストア](#)を作成するオプションがあります。Verified Permissions は、リクエスト Authorization ヘッダーから ID トークンまたはアクセストークンを処理する Lambda オーソライザーを作成して割り当てます。この Lambda オーソライザーは、トークンをポリシーストアに渡します。ここで、Verified Permissions はトークンをポリシーと比較し、オーソライザーに許可または拒否の決定を返します。

その他のリソース

- [API Gateway での REST API へのアクセスの制御と管理](#)
- [Amazon Verified Permissions による承認](#)

Machine to machine (M2M) 認可

Amazon Cognito は、マシン ID を使用して API データにアクセスするアプリケーションをサポートしています。ユーザープール内のマシン ID は、アプリケーションサーバー上で実行され、リモート

API に接続する [機密クライアント](#) です。オペレーションは、スケジュールされたタスク、データストリーム、アセットの更新など、ユーザーとのインタラクションなしで行われます。これらのクライアントは、アクセストークンを使用してリクエストを認可すると、Machine to Machine (M2M) 認可を実行します。M2M 認可では、共有シークレットがアクセスコントロールのユーザー認証情報を置き換えます。

M2M 認可で API にアクセスするアプリケーションには、クライアント ID とクライアントシークレットが必要です。ユーザープールでは、クライアント認証情報の付与をサポートするアプリケーションクライアントを構築する必要があります。クライアント認証情報をサポートするには、アプリケーションクライアントにクライアントシークレットが必要であり、ユーザープールドメインがなくてはなりません。このフローでは、マシン ID は [トークンエンドポイント](#) から直接アクセストークンをリクエストします。クライアント認証情報の付与には、アクセストークン内の [リソースサーバー](#) からのカスタムスコープのみを認可できます。アプリケーションクライアントのセットアップの詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

クライアント認証情報の付与からのアクセストークンは、マシン ID が API にリクエストすることを許可するオペレーションの検証可能なステートメントです。アクセストークンが API リクエストを認可する方法の詳細については、引き続きお読みください。アプリケーションの例については、「[Amazon Cognito and API Gateway based machine to machine authorization using AWS CDK](#)」を参照してください。

M2M 認可には、月間アクティブユーザー (MAU) に請求する方法とは異なる請求モデルがあります。ユーザー認証にアクティブなユーザーあたりのコストがかかる場合、M2M 請求にはアクティブなクライアント認証情報アプリケーションクライアントと、トークンリクエストの合計ボリュームが反映されます。詳細については、「[Amazon Cognito の料金](#)」を参照してください。M2M 認可のコストを制御するには、アクセストークンの期間と、アプリケーションが行うトークンリクエストの数を最適化します。API Gateway キャッシングを使用して M2M 認可の新しいトークンのリクエストを減らす方法については、「[ユーザープールのトークンの有効期限とキャッシュの管理](#)」を参照してください。

AWS 請求書にコストを追加する Amazon Cognito オペレーションの最適化については、「」を参照してください [のコスト管理](#)。

Machine to Machine (M2M) クライアント認証情報のクライアントメタデータ

[クライアントメタデータ](#) は M2M リクエストで渡すことができます。クライアントメタデータは、[トークン生成前の Lambda トリガー](#) の結果に役立つユーザーまたはアプリケーション環境からの追加情報です。ユーザープリンシパルによる認証オペレーションで

は、[AdminRespondToAuthChallenge](#) API リクエストおよび [RespondToAuthChallenge](#) API リクエストの本文で、クライアントメタデータをトークン生成前トリガーに渡すことができます。アプリケーションは、[トークンエンドポイント](#) への直接リクエストを使用して M2M のアクセストークン生成フローを実行するため、異なるモデルとなります。クライアント認証情報のトークンリクエストの POST 本文で、クライアントメタデータオブジェクトを URL エンコード (x-www-form-urlencoded) した `aws_client_metadata` パラメータを文字列に渡します。リクエスト例については、「[基本認可によるクライアント認証情報](#)」を参照してください。次に示すのは、キーと値のペア `{"environment": "dev", "language": "en-US"}` を渡すパラメータの例です。

```
aws_client_metadata=%7B%22environment%22%3A%20%22dev%22,%20%22language%22%3A%20%22en-US%22%7D
```

スコープについて

スコープはアプリがリソースにリクエストできるアクセスのレベルです。Amazon Cognito アクセストークンのスコープは、ユーザープールで設定した信頼 (既知のデジタル署名を持つアクセストークンの信頼できる発行者) によってバックアップされます。ユーザープールが生成できるアクセストークンのスコープでは、顧客が自分のユーザープロフィールの一部または全部の管理や、バックエンド API からのデータの取得を許可されていることを証明します。Amazon Cognito ユーザープールが発行するアクセストークンのスコープには、ユーザープールのリザーブド API スコープ、カスタムスコープ、および OpenID Connect (OIDC) スコープがあります。

ユーザープールのリザーブド API スコープ

`aws.cognito.signin.user.admin` スコープは、Amazon Cognito ユーザープール API の現在のユーザーのセルフサービスオペレーションを認可します。アクセストークンのベアラーに対して、当該ベアラーに関するすべての情報を、[GetUser](#) や [UpdateUserAttributes](#) などの API オペレーションを使用してクエリおよび更新することを認可します。Amazon Cognito ユーザープール API でユーザーを認証する場合は、このスコープだけをアクセストークンで受け取ります。また、アプリケーションクライアントにユーザー属性の読み取りと書き込みを許可している場合、このユーザー属性の読み取りと書き込みに必要な唯一のスコープでもあります。このスコープは、[認可エンドポイント](#) へのリクエストでリクエストすることもできます。このスコープだけでは、[userInfo エンドポイント](#) にユーザー属性をリクエストするには不十分です。ユーザープール API とユーザーへの `userInfo` リクエストの両方を承認するアクセストークンについては、`/oauth2/authorize` リクエストで `openid` スコープと `aws.cognito.signin.user.admin` スコープの両方をリクエストする必要があります。

カスタムスコープ

カスタムスコープは、リソースサーバーで保護する外部 API へのリクエストを承認します。カスタムスコープを他の種類のスコープと一緒にリクエストできます。カスタムスコープの詳細については、このページ全体を通じて説明しています。

OpenID Connect (OIDC) のスコープ

ユーザープール認可サーバー (マネージドログインを含む) でユーザーを認証する場合は、スコープをリクエストする必要があります。ユーザープールのローカルユーザーとサードパーティのフェデレーションユーザーは、Amazon Cognito 認可サーバーで認証できます。OIDC スコープは、ユーザープールの [userInfo エンドポイント](#) からユーザー情報を読み取ることをアプリケーションに認可します。userInfo エンドポイントからユーザー属性をクエリする OAuth モデルでは、ユーザー属性に対する大量のリクエストに合わせてアプリケーションを最適化できます。userInfo エンドポイントは、アクセストークンのスコープによって決まるアクセス許可レベルで属性を返します。以下の OIDC スコープでアクセストークンを発行することをアプリケーションクライアントに認可できます。

openid

OpenID Connect (OIDC) クエリの最小スコープ。ID トークン、一意の識別子のクレーム sub、および他のスコープをリクエストする機能を承認します。

Note

openid スコープをリクエストし、それ以外をリクエストしない場合、ユーザープール ID トークンと userInfo レスポンスには、アプリケーションクライアントが読み取ることができるすべてのユーザー属性のクレームが含まれます。openid や他の OIDC スコープ (profile、email、phone など) をリクエストすると、ID トークンと [userInfo](#) レスポンスの内容は追加のスコープの制約に制限されます。

例えば、[認可エンドポイント](#) へのリクエストでパラメータ scope=openid+email を使用すると、sub、email、email_verified を含む ID トークンが返されます。このリクエストのアクセストークンは、[userInfo エンドポイント](#) から同じ属性を返します。リクエストでパラメータ scope=openid を使用すると、ID トークンと userInfo 内のクライアントが読み取り可能なすべての属性が返されます。

profile

アプリケーションクライアントが読み取ることができるすべてのユーザー属性を承認します。

email

ユーザー属性 email および email_verified を承認します。Amazon Cognito は、値が明示的に設定されている場合、email_verified を返します。

phone

ユーザー属性 phone_number および phone_number_verified を承認します。

リソースサーバーについて

リソースサーバー API は、データベース内の情報へのアクセスを許可したり、IT リソースを制御したりすることができます。Amazon Cognito のアクセストークンは、OAuth 2.0 をサポートする API へのアクセスを許可できます。Amazon API Gateway REST API には、Amazon Cognito のアクセストークンによる認可の[組み込みサポート](#)があります。アプリは、API コール内でアクセストークンをリソースサーバーに渡します。リソースサーバーはアクセストークンを検査し、アクセスを付与するかどうかを決定します。

Amazon Cognito は、ユーザープールのアクセストークンのスキーマを将来更新する可能性があります。アプリケーションがアクセストークンを API に渡す前に、アクセストークンの内容を分析する場合は、スキーマの更新を受け入れるようにコードを設計する必要があります。

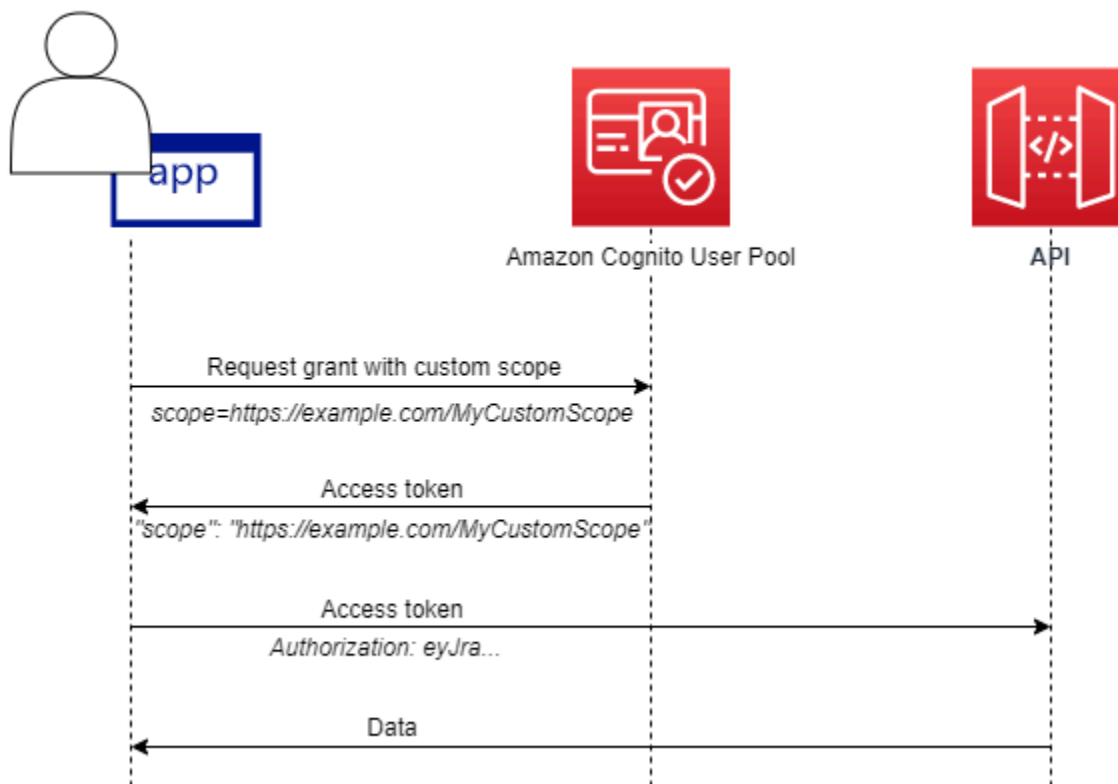
カスタムスコープはユーザーが定義するスコープであり、ユーザープールの承認機能を拡張して、ユーザーとユーザー属性のクエリや変更とは関係のない目的にも対応できるようにします。例えば、写真用のサーバーリソースがある場合、2 つのスコープを定義することが考えられます。写真への読み取りアクセス用の photos.read と、書き込み/削除アクセス用の photos.write です。アクセストークンを承認して許可するように API を設定し、scope クレームでの photos.read によるアクセストークンへの HTTP GET リクエスト、および photos.write によるトークンへの HTTP POST リクエストを許可できます。これらはカスタムスコープです。

Note

リソースサーバーはトークン内のクレームを処理する前に、アクセストークンの署名と有効期限を検証する必要があります。トークンの検証の詳細については、「[JSON ウェブトークンの検証](#)」を参照してください。Amazon API Gateway でユーザープールのトークンを検証および使用方法の詳細については、ブログ「[Amazon Cognito ユーザープールと API Gateway の統合](#)」を参照してください。API Gateway は、アクセストークンの検証とリソースの保護に適したオプションです。API Gateway カスタムオーソライザーの詳細については、「[API Gateway Lambda オーソライザーを使用する](#)」を参照してください。

概要:

Amazon Cognito を使用すると、OAuth2.0 リソースサーバーを作成し、これらにカスタムスコープを関連付けることができます。アクセストークンのカスタムスコープは、API の特定のアクションを許可します。ユーザープール内のどのアプリケーションクライアントに対しても、任意のリソースサーバーからカスタムスコープを発行することを許可できます。カスタムスコープをアプリケーションクライアントに関連付け、これらのスコープを [トークンエンドポイント](#) から OAuth2.0 認可コード付与、暗黙的な付与、クライアント認証情報付与でリクエストできます。Amazon Cognito は、アクセストークンの scope クレームにカスタムスコープを追加します。クライアントはサーバーリソースに対してアクセストークンを使用して、トークンに存在するスコープに基づいて承認を判断できます。アクセストークンスコープの詳細については、「[ユーザープールのトークンの使用](#)」を参照してください。



カスタムスコープでアクセストークンを取得するには、アプリケーションは認可コードを引き換えるか、クライアント認証情報の付与をリクエストするために、[トークンエンドポイント](#) にリクエストを行う必要があります。マネージドログインでは、暗黙的な付与からアクセストークン内のカスタムスコープをリクエストすることもできます。

Note

[InitiateAuth](#) リクエストと [AdminInitiateAuth](#) リクエストは、ユーザープールを IdP として人間との対話型の認証を行うように設計されているため、アクセストークンの scope クレームを単一の値 `aws.cognito.signin.user.admin` で生成するだけです。

リソースサーバーおよびカスタムスコープの管理

リソースを作成するときに、リソースサーバー名とリソースサーバー識別子を指定する必要があります。リソースサーバーに作成したスコープごとに、スコープ名と説明を指定する必要があります。

- リソースサーバー名: リソースサーバーのわかりやすい名前 (Solar system object tracker や Photo API など)。
- リソースサーバーの識別子: リソースサーバーの一意の識別子。識別子は、API に関連付ける任意の名前 (solar-system-data など) です。API URI パスへの直接参照のように長い識別子 (`https://solar-system-data-api.example.com` など) を設定できますが、文字列が長くなるとアクセストークンのサイズが大きくなります。
- スコープ名: scope クレームに必要な値。例えば、`sunproximity.read`。
- 説明: スコープのわかりやすい説明。例えば、`Check current proximity to sun`。

Amazon Cognito では、ユーザープールのローカルユーザーであるか、サードパーティ ID プロバイダーのフェデレーションユーザーであるかに関係なく、すべてのユーザーのアクセストークンにカスタムスコープを含めることができます。マネージドログインを含む OAuth 2.0 認可サーバーによる認証フロー中に、ユーザーのアクセストークンのスコープを選択できます。ユーザーの認証は、リクエストパラメータの 1 つとして scope を使用して [認可エンドポイント](#) で開始する必要があります。リソースサーバーには次の形式が推奨されます。識別子として、API のわかりやすい名前を使用します。カスタムスコープには、リソースサーバーが許可するアクションを使用します。

`resourceServerIdentifier/scopeName`

例えば、カイパーベルトで新しい小惑星を発見し、それを solar-system-data API を通じて登録するとします。小惑星のデータベースへの書き込み操作を許可するスコープは `asteroids.add` です。発見の登録を許可するアクセストークンをリクエストするときは、scope HTTPS リクエストパラメータを `scope=solar-system-data/asteroids.add` としてフォーマットします。

スコープをリソースサーバーから削除しても、すべてのクライアントとの関連付けは削除されません。代わりに、スコープが `inactive` とマークされます。Amazon Cognito はアクセストークンに非アクティブなスコープを追加しませんが、それ以外はアプリからのスコープのリクエストを通常どおりに処理します。後でスコープをリソースサーバーに再度追加すると、Amazon Cognito はそのスコープを再度アクセストークンに書き込みます。アプリケーションクライアントに関連付けていないスコープをリクエストすると、ユーザープールのリソースサーバーからスコープを削除したかどうかに関係なく、認証は失敗します。

AWS マネジメントコンソール、API、または CLI を使用して、ユーザープールのリソースサーバーとスコープを定義できます。

ユーザープールのリソースサーバーを定義する (AWS マネジメントコンソール)

を使用して AWS マネジメントコンソール、ユーザープールのリソースサーバーを定義できます。

リソースサーバーを定義する

1. [Amazon Cognito コンソール](#) にサインインします。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [ブランディング] の下で [ドメイン] メニューを選択し、[リソースサーバー] を見つけます。
4. [Create a resource server] (リソースサーバーの作成) を選択します。
5. [Resource server name] (リソースサーバー名) を入力します。例えば、Photo Server。
6. [Resource server identifier] (リソースサーバー識別子) を入力します。例えば、com.example.photos。
7. リソースの [Custom scopes] (カスタムスコープ) (例: read および write) を入力します。
8. [Scope name] (スコープ名) ごとに [Description] (説明) (例: view your photos および update your photos) を入力します。
9. [作成] を選択します。

カスタムスコープは、[ドメイン] メニューで [リソースサーバー] の [カスタムスコープ] 列で確認できます。カスタムスコープは、アプリケーションクライアントに対して [アプリケーション] の [アプリケーションクライアント] で有効にすることができます。アプリケーションクライアントを選択し、[ログインページ] を見つけて [編集] を選択します。[Custom scopes] (カスタムスコープ) を追加し、[Save changes] (変更の保存) を選択します。

ユーザープール (AWS CLI および AWS API) のリソースサーバーの定義

次のコマンドを使用して、ユーザープールのリソースサーバー設定を指定します。

リソースサーバーを作成する

- AWS CLI: `aws cognito-idp create-resource-server`
- AWS API: [CreateResourceServer](#)

リソースサーバーの設定に関する情報を取得する

- AWS CLI: `aws cognito-idp describe-resource-server`
- AWS API: [DescribeResourceServer](#)

ユーザープールのすべてのリソースサーバーに関する情報を一覧表示する

- AWS CLI: `aws cognito-idp list-resource-servers`
- AWS API: [ListResourceServers](#)

リソースサーバーを削除するには

- AWS CLI: `aws cognito-idp delete-resource-server`
- AWS API: [DeleteResourceServer](#)

リソースサーバーの設定を更新する

- AWS CLI: `aws cognito-idp update-resource-server`
- AWS API: [UpdateResourceServer](#)

リソースバインディング

リソースバインディング (リソースインジケーターとも呼ばれます) を使用すると、ユーザープールの認可サーバーに対して API 固有の付与をリクエストできます。リソースバインディングは、[RFC 8707](#) に定義されている OAuth 2.0 の拡張機能です。これにより、クライアントは認可リクエスト時に、どのリソースサーバーにアクセスするかを明示的に指定できます。リソースバインディングを使用すると、API 設定では、特に意図されていないトークンへのアクセスを拒否できます。

Note

アクセストークンは、ユーザーのリソースにのみバインドできます。クライアント認証情報の M2M 付与を使用してリソースバインディングをリクエストすることはできません。

Amazon Cognito ユーザープールでリソースバインディングを使用する場合、クライアントはユーザープール認可サーバーに対する認証リクエストに `resource` パラメータを含めることができます。ユーザープールは、リクエストされたリソースの値が URL であり、[アプリケーションクライアント](#)のコールバック URL (localhost 専用の `https://` と `http://`、または `myapp://` などのカスタムスキーム) と同じスキームルールに従っていることを検証します。Amazon Cognito は、[アクセストークン](#)の `aud` クレームで、リクエストされた URI をオーディエンスとして設定します。リクエストされたリソースがユーザープールのリソースサーバーである場合、リソースサーバーの識別子は URL 形式である必要があります。認証リクエストごとに 1 つのリソースをリクエストできます。

この機能は、ユーザープールの OAuth 2.0 認可サーバーによる[マネージドログイン認証](#)専用です。リソースバインディングは、[認可エンドポイント](#)からの暗黙的な付与および認証コード付与でリクエストできます。[トークンエンドポイント](#)からのトークン更新の付与は、元のリクエストから `aud` クレームを継承します。現在、[SDK 認証モデル](#)では使用できません。

Amazon Cognito ユーザープールでリソースバインディングを実装する

1. 一意の識別子を使用して、ユーザープール内の 1 つ以上のリソースサーバーを設定します。
2. `/oauth2/authorize` への認可リクエストでは、認可コードまたは暗黙的な付与をリクエストし、`resource` パラメータを含めます。`resource` の値は、URL 形式のリソースサーバー識別子または URL である必要があります。例えば、`&resource=https://solar-system-data-api.example.com`。
3. 認可サーバーは、リソースリクエストを検証して認証を完了し、アクセストークンの `aud` クレームを、リクエストされたリソース URL に設定します。
4. トークンがリクエスト先に発行されたことを検証するために、ユーザーのアクセストークンを使用するリソースは、`aud` クレームをチェックします。

ユーザープールの機能を設定する

これまでの章では、Amazon Cognito コンソールからのガイダンスに従っていくつかの機能を設定してみました。このセクションのページでは、ユーザープールの主要機能の一部について、設定要件を

より掘り下げます。アプリケーションクライアント、E メールと SMS の設定、ユーザーデバイスの記憶などのオプションについて、重要なリファレンス情報があります。

トピック

- [ユーザープールとアプリケーションクライアントの設定更新](#)
- [アプリケーションクライアントによるアプリケーション固有の設定](#)
- [ユーザープール内のユーザーデバイスの使用](#)
- [ユーザープール分析に Amazon Pinpoint を使用する](#)
- [Amazon Cognito ユーザープールの E メール設定](#)
- [Amazon Cognito ユーザープール用の SMS メッセージ設定](#)

ユーザープールとアプリケーションクライアントの設定更新

ユーザープールまたはアプリケーションクライアントの設定を変更する場合は、Amazon Cognito コンソールで数回クリックすれば更新を適用できます。ユーザープール設定の機能ベースのタブを移動して、このガイドの他のエリアの説明に従ってフィールドを更新します。

多くの組織は、のリソース AWS CloudFormation、AWS SDKs または CDK 上に構築されたアプリケーション、およびその他のオートメーションソフトウェアをプログラムで管理しています。これがリソース管理モデルである場合は、リソースの変更を段階的に行うときに特に注意する必要があります。

API オペレーションの [UpdateUserPool](#) および [UpdateUserPoolClient](#) は、既存のユーザープールまたはアプリケーションクライアントを更新します。それぞれに、次の API リファレンスの警告が表示されます: 属性の値を指定しない場合、Amazon Cognito はそれをデフォルト値に設定します。1 つのパラメータのみを使用して更新リクエストを送信すると、Amazon Cognito はそのパラメータを選択した値に設定し、他のすべてのパラメータをデフォルト値に設定します。これにより、属性スキーマ、Lambda トリガー、E メールおよび SMS メッセージ設定などの設定をリセットできます。

さらに、ユーザープールまたはアプリケーションクライアントの作成後に一部の設定がロックされ、新しいリソースを作成しない限り変更することはできません。

トピック

- [変更できない設定](#)
- [SMS 設定](#)
- [AWS SDK、AWS CDK、または REST API を使用したユーザープールの更新](#)

変更できない設定

ユーザープールを作成した後に、この設定を変更することはできません。以下の設定を変更する場合は、新しいユーザープールまたはアプリケーションを作成する必要があります。

Note

以前は、ユーザープールの名前を変更できませんでした。これは変更されました。ユーザープールに新しいわかりやすい名前を割り当てることができるようになりました。

ユーザープール ID

API パラメータ名: [Id/UserPoolId](#)

ユーザープール ID (us-east-1_EXAMPLE など) は、Amazon Cognito によって自動生成され、変更できません。

Amazon Cognito ユーザープールサインインオプション

API パラメータ名: [AliasAttributes](#) と [UsernameAttributes](#)

ユーザーがサインインするときにユーザー名として渡すことができる属性。ユーザープールを作成するときに、ユーザー名、E メールアドレス、電話番号、または優先ユーザー名を使用してサインインを許可するかどうかを選択できます。ユーザープールサインインオプションを変更するには、新しいユーザープールを作成します。

ユーザー名の大文字と小文字を区別する

API パラメータ名: [UsernameConfiguration](#)

大文字と小文字以外の別のユーザー名と一致するユーザー名を作成した場合、Amazon Cognito では同じユーザーまたは一意のユーザーとして扱うことができます。詳細については、「[ユーザープールの大文字と小文字の区別](#)」を参照してください。大文字と小文字の区別を変更するには、新しいユーザープールを作成します。

クライアントシークレット

API パラメータ名: [GenerateSecret](#)

アプリケーションを作成するときに、信頼されたソースだけがユーザープールにリクエストを送信できるように、クライアントシークレットを生成できます。詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。クライアント

シークレットを変更するには、同じユーザープール内に新しいアプリケーションを作成します。

必須の属性

API パラメータ名: [Schema](#)

ユーザーのサインアップ時または作成時に値を提供する必要がある属性。詳細については、「[ユーザー属性の操作](#)」を参照してください。必要な属性を変更するには、新しいユーザープールを作成します。

カスタム属性 (削除)

API パラメータ名: [Schema](#)

カスタム名を持つ属性です。ユーザーのカスタム属性の値を変更することはできますが、ユーザープールからカスタム属性を削除することはできません。詳細については、「[ユーザー属性の操作](#)」を参照してください。カスタム属性の最大数に達してリストを変更する場合は、新しいユーザープールを作成します。

SMS 設定

ユーザープールで SMS メッセージをアクティブ化した後は、それらを非アクティブ化できません。

- ユーザープールの作成時に SMS メッセージを設定することを選択した場合、セットアップ完了後に SMS を非アクティブ化できません。
- 作成したユーザープールで SMS メッセージをアクティブ化できますが、その後は SMS を非アクティブ化できません。
- Amazon Cognito では、ユーザーアカウントの招待と復旧、属性の検証、多要素認証 (MFA) に SMS メッセージを使用できます。SMS メッセージをアクティブ化した後は、いつでも SMS メッセージでこれらの機能をオンまたはオフにできます。
- SMS メッセージ設定には、Amazon SNS によるメッセージ送信を Amazon Cognito に委任する IAM ロールが含まれています。割り当てられたロールはいつでも変更できます。

AWS SDK、AWS CDK、または REST API を使用したユーザープールの更新

Amazon Cognito コンソールでは、ユーザープールの設定を 1 つのパラメータずつ変更できます。例えば、Lambda トリガーを追加するには、[Lambda トリガーを追加] を選択し、関数とトリガーのタイプを選択します。Amazon Cognito ユーザープール API は、ユーザープールとアプリケーションクライアントの更新オペレーションに、ユーザープールのパラメーター式が必要となるように構成され

ています。ただし、コンソールは他のユーザープール設定を使用して、この更新オペレーションを透過的に自動化します。

他の場所での変更によって、変更する設定に関連しない AWS アカウント 更新でエラーが発生することがあります。例えば、削除された Amazon SES ID AWS WAF や IAM アクセス許可の変更。現在のパラメータの 1 つが無効になった場合、それを修正するまで設定を更新できません。このようなエラーが発生した場合は、エラー応答を調べ、言及されている設定を検証します。

[AWS Cloud Development Kit \(AWS CDK\)](#)、[Amazon Cognito ユーザープール REST API](#) および [AWS SDK](#) は、Amazon Cognito リソースの自動化とプログラムによる設定を行うためのツールです。これらのツールを使用したリクエストは、Amazon Cognito コンソールと同様に、リクエスト本文の完全なリソース設定で設定の更新を行う必要もあります。概要としては、以下のプロセスを実行する必要があります。

1. 既存のリソースの設定を記述するオペレーションから出力をキャプチャします。
2. 設定を変更して出力を変更します。
3. リソースを更新するオペレーションで、変更された設定を送信します。

次の手順で、[UpdateUserPool](#) API オペレーションを使用して設定を更新します。入力フィールドが異なる同じアプローチが、[UpdateUserPoolClient](#) に適用されます。

Important

既存のパラメータの値を提供しない場合、Amazon Cognito はそれらをデフォルト値に設定します。例えば、既存の LambdaConfig がある場合に、空の LambdaConfig で UpdateUserPool を送信すると、すべての Lambda 関数のユーザープールトリガーへの割り当てが削除されます。ユーザープール設定の変更を自動化する場合は、適宜計画を立ててください。

1. [DescribeUserPool](#) を使用して、ユーザープールの既存の状態をキャプチャします。
2. DescribeUserPool の出力を UpdateUserPool の [リクエストパラメータ](#) と一致するようにフォーマットします。出力される JSON から次のトップレベルフィールドとその子オブジェクトを削除します。
 - Arn
 - CreationDate

- CustomDomain
 - このフィールドを [UpdateUserPoolDomain](#) API オペレーションを使用して更新します。
 - Domain
 - このフィールドを [UpdateUserPoolDomain](#) API オペレーションを使用して更新します。
 - EmailConfigurationFailure
 - EstimatedNumberOfUsers
 - Id
 - LastModifiedDate
 - Name
 - SchemaAttributes
 - SmsConfigurationFailure
 - Status
3. 生成された JSON が UpdateUserPool の [リクエストパラメータ](#) と一致することを確認します。
 4. 生成された JSON で変更するパラメータを変更します。
 5. 変更した JSON をリクエスト入力として使用する UpdateUserPool API リクエストを送信します。

また、AWS CLI の `update-user-pool` の `--cli-input-json` パラメータで、この修正した `DescribeUserPool` の出力を使用することもできます。

または、次の AWS CLI コマンドを実行して、で受け入れられる入力フィールドの値が空白の JSON を生成します `update-user-pool`。その後、これらのフィールドにユーザープールの既存の値を入力できます。

```
aws cognito-idp update-user-pool --generate-cli-skeleton --output json
```

次のコマンドを実行して、アプリケーションクライアント用に同じ JSON オブジェクトを生成します。

```
aws cognito-idp update-user-pool-client --generate-cli-skeleton --output json
```

アプリケーションクライアントによるアプリケーション固有の設定

ユーザープールアプリケーションクライアントは、Amazon Cognito で認証される 1 つのモバイルアプリケーションまたはウェブアプリケーションを操作するユーザープール内の設定です。アプリケーションクライアントは、認証済みおよび未認証の API オペレーションを呼び出したり、ユーザーの属性の一部またはすべてを読み取ったり変更したりできます。アプリでは、登録、サインイン、パスワードを忘れた場合の処理を行う際に、アプリケーションクライアントに対して自分自身を識別する必要があります。これらの API リクエストには、アプリケーションクライアント ID による自己識別と、オプションのクライアントシークレットによる認可が含まれていなければなりません。認証されたクライアントアプリのみがこれらの認証されていない API を呼び出すことができるように、アプリケーションクライアント ID またはシークレットをセキュア化する必要があります。さらに、認証された API リクエストに AWS 認証情報で署名するようにアプリを設定する場合は、認証情報をユーザー検査から保護する必要があります。

ユーザープールには複数のアプリを作成できます。アプリケーションクライアントは、アプリのコードプラットフォームにリンクされている場合もあれば、ユーザープール内の別のテナントにリンクされている場合もあります。例えば、サーバー側アプリケーション用のアプリと、別の Android アプリを作成することができます。各アプリには独自のアプリのクライアント ID があります。

アプリケーションクライアントレベルで、次のユーザープール機能の設定を適用できます。

1. [分析](#)
2. [マネージドログイン](#) の IdP、付与タイプ、コールバック URL、およびカスタマイズ
3. [リソースサーバーとカスタムスコープ](#)
4. [脅威保護](#)
5. [属性の読み取り/書き込み許可](#)
6. [トークンの有効期限と取り消し](#)
7. [認証フロー](#)

アプリケーションクライアントの種類

Amazon Cognito でアプリケーションクライアントを作成する際、標準の OAuth クライアントタイプであるパブリッククライアントと機密クライアントに基づいたオプションを事前に入力できます。クライアントシークレットを持つ機密クライアントを設定します。クライアントの種類の詳細については、[IETF RFC 6749 #2.1](#) を参照してください。

パブリッククライアント

パブリッククライアントは、ブラウザまたはモバイルデバイスで実行されます。信頼できるサーバー側リソースがないため、クライアントシークレットはありません。

機密クライアント

機密クライアントには、認証されていない API オペレーションのクライアントシークレットで信頼できるサーバー側のリソースがあります。アプリケーションは、バックエンドサーバー上でデーモンまたはシェルスクリプトとして実行されることがあります。

クライアントシークレット

クライアントシークレットまたはクライアントパスワードは、アプリクライアントへのすべての API リクエストでアプリが使用する必要がある固定文字列です。アプリクライアントには、`client_credentials` 付与を実行するためのクライアントシークレットが必要です。詳細については、[IETF RFC 6749 #2.3.1](#) を参照してください。

各アプリケーションクライアントは一度に最大 2 つのシークレットを持つことができ、ダウンタイムなしでシークレットをローテーションできます。アプリケーションクライアントを作成するときに、Amazon Cognito にシークレット値を生成させるか、独自のカスタムシークレット値を提供させることができます。アプリを作成した後は、シークレットを変更できません。[AddUserPoolClientSecret](#) API オペレーションを使用して 2 番目のシークレットを追加して、シークレットをローテーションできます。シークレットを追加するときは、Amazon Cognito にシークレット値を生成させるか、独自のカスタムシークレット値を提供させることができます。シークレットを削除するには、[DeleteUserPoolClientSecret](#) API オペレーションを使用します。アプリケーションクライアントに関連付けられた唯一のシークレットを削除することはできません。また、そのアプリのクライアント ID を使用するアプリケーションからのアクセスを遮断するためにアプリケーションを削除できます。

Note

アプリケーションタイプとして [従来のウェブアプリケーション] オプションと [Machine to Machine アプリケーション] オプションを選択すると、Amazon Cognito コンソールはクライアントシークレットを持つアプリケーションクライアントを作成します。これらのオプションのいずれかを選択してクライアントシークレットを生成するか、[CreateUserPoolClient](#) を使用してプログラムでクライアントを作成して `GenerateSecret` を `true` に設定します。

パブリックアプリでは、機密クライアントとクライアントシークレットを使用できます。Amazon CloudFront プロキシを使用して、転送中の SECRET_HASH を追加します。詳細については、AWS ブログの「[Amazon CloudFront プロキシを使用して Amazon Cognito のパブリッククライアントを保護する](#)」を参照してください。

JSON ウェブトークン

Amazon Cognito アプリクライアントは、以下のタイプの JSON ウェブトークン (JWT) を発行できます。

アイデンティティ (ID) トークン

ユーザーがユーザープールから認証されていることを示す検証可能なステートメント。OpenID Connect (OIDC) は、OAuth 2.0 で定義されているアクセストークンとリフレッシュトークンの標準に [ID トークンの仕様](#)を追加しました。ID トークンには、アプリがユーザープロフィールの作成やリソースのプロビジョニングに使用できるユーザー属性などの ID 情報が含まれています。詳細については「[ID トークンの理解](#)」を参照してください。

アクセストークン

ユーザーのアクセス権を示す検証可能なステートメント。アクセストークンには、OIDC と OAuth 2.0 の機能である [スコープ](#)が含まれています。アプリは、バックエンドリソースにスコープを提示することで、ユーザープールが API のデータや独自のユーザーデータへのアクセスをユーザーまたはマシンに許可していることを証明できます。カスタムスコープを持つアクセストークンは、多くの場合、M2M クライアント認証情報の付与により、リソースサーバーへのアクセスを許可します。詳細については「[アクセストークンの理解](#)」を参照してください。

更新トークン

ユーザーのトークンが有効期限切れになったときにアプリがユーザープールに提示できる、暗号化された初期認証ステートメント。更新トークンをリクエストすると、有効期限が切れていない新しいアクセストークンと ID トークンが返されます。詳細については「[更新トークン](#)」を参照してください。

各アプリケーションクライアントのこれらのトークンの有効期限は、[Amazon Cognito コンソール](#)でユーザープールの [アプリケーションクライアント] メニューから設定できます。

アプリクライアントの用語

Amazon Cognito コンソールで、アプリクライアントに使用できる用語は次のとおりです。

許可されているコールバック URL

コールバック URL は、ユーザーがサインインに成功したときにリダイレクトされる先を指定します。少なくとも 1 つのコールバック URL を選択してください。コールバック URL は、以下が必要です。

- 絶対 URI である。
- クライアントに事前登録されている。
- フラグメントコンポーネントを含まない。

「[OAuth 2.0 - リダイレクトエンドポイント](#)」を参照してください。

Amazon Cognito は、テスト目的限定の HTTPS を除き、HTTP ではなく `http://localhost` を使用します。

また、アプリのコールバック URL (例: `myapp://example`) もサポートされています。

許可されているサインアウト URL

サインアウト URL は、ユーザーがサインアウトしたときにリダイレクトされる先を指定します。

属性の読み取りおよび書き込み許可

ユーザープールには多数の顧客がいて、それぞれが独自のアプリクライアントと IdP を使用している場合があります。アプリに関連したユーザー属性のみの読み取りおよび書き込みアクセスを許可するようにアプリクライアントに設定できます。マシンツーマシン (M2M) 認可のように、すべてのユーザー属性へのアクセスを禁止することもできます。

属性の読み取りおよび書き込みアクセス権限の設定に関する考慮事項

- アプリケーションクライアントを作成する場合で、属性の読み取り/書き込みアクセス許可をカスタマイズしない場合、Amazon Cognito はすべてのユーザープール属性に読み取り/書き込みアクセス許可を付与します。
- イミュータブルな [カスタム属性](#) への書き込みアクセス権を付与できます。ユーザーの作成時またはサインアップ時に、アプリケーションクライアントはイミュータブルな属性に値を書き込むことができます。その後は、ユーザーのイミュータブルなカスタム属性に値を書き込むことはできません。
- アプリクライアントは、ユーザープール内の必須属性への書き込み権限を持っている必要があります。Amazon Cognito コンソールは、必須の属性を自動的に書き込み可能に設定します。

- アプリケーションクライアントに対して、`email_verified` または `phone_number_verified` への書き込みアクセス権限を付与することはできません。ユーザープール管理者は、これらの値を変更できます。ユーザーは、これらの属性の値を[属性の検証](#)でのみ変更できます。

認証フロー

アプリクライアントが許可するサインイン方法。アプリケーションは、ユーザー名とパスワードによる認証、E メールおよび SMS メッセージの OTP、パスキー認証、Lambda トリガーによるカスタム認証、トークン更新をサポートできます。セキュリティのベストプラクティスとして、カスタム構築されたアプリケーションでのユーザー名とパスワード認証には SRP 認証を使用します。

カスタムスコープ

カスタムスコープは、[リソースサーバー] で各自のリソースサーバー用に定義したものです。形式は `resource-server-identifier/scope` です。「[スコープ、M2M、リソースサーバー](#)」を参照してください。

デフォルトのリダイレクト URI

サードパーティー IdP を持つユーザーの認証リクエストの `redirect_uri` パラメータを置き換えます。[CreateUserPoolClient](#) API リクエストまたは [UpdateUserPoolClient](#) API リクエストの `DefaultRedirectURI` パラメータを使用して、このアプリケーションクライアント設定をセットアップします。また、この URL は、アプリケーションクライアント用の CallbackURLs のメンバーである必要もあります。Amazon Cognito は、次の場合に、認証されたセッションをこの URL にリダイレクトします。

1. アプリケーションクライアントには、1 つの [ID プロバイダー](#) が割り当てられ、複数の [コールバック URL](#) が定義されています。ユーザープールは、認証リクエストに `redirect_uri` パラメータが含まれていない場合、[認可サーバー](#) への認証リクエストをデフォルトのリダイレクト URI にリダイレクトします。
2. アプリケーションクライアントには、1 つの [ID プロバイダー](#) が割り当てられ、1 つの [コールバック URL](#) が定義されています。このシナリオでは、デフォルトのコールバック URL を定義する必要はありません。`redirect_uri` パラメータを含まないリクエストは、使用可能なコールバック URL の 1 つにリダイレクトされます。

ID プロバイダ

ユーザープールの外部 ID プロバイダー (IdP) の一部またはすべてを選択して、ユーザーを認証できます。アプリクライアントは、ユーザープール内のローカルユーザーのみを認証することもで

きます。IdP をアプリケーションクライアントに追加すると、IdP への認可リンクを生成して、マネージドログインのサインインページで表示できます。複数の IdP を割り当てることができ、少なくとも 1 つを割り当てる必要があります。外部 IdP の使用の詳細については、「[サードパーティーの ID プロバイダーを使用したユーザープールへのサインイン](#)」を参照してください。

OpenID Connect スコープ

次の OAuth スコープを 1 つ以上選択し、アクセストークン用にリクエストできるアクセス権限を指定します。

- openid スコープでは、ID トークンとユーザーの固有 ID を取得することを宣言します。また、リクエストに含まれる追加のスコープに応じて、すべてまたは一部のユーザー属性もリクエストします。openid スコープをリクエストしない限り、Amazon Cognito は ID トークンを返しません。openid スコープは、有効期限やキー ID などの構造的な ID トークンのクレームを承認し、[userInfo エンドポイント](#) からのレスポンスで受け取るユーザー属性を決定します。
- リクエストするスコープが openid のみである場合、Amazon Cognito は現在のアプリケーションクライアントが読み取ることができるすべてのユーザー属性を ID トークンに入力します。このスコープだけを使用したアクセストークンへの userInfo レスポンスにより、すべてのユーザー属性が返されます。
- phone、email、profile などの他のスコープで openid をリクエストすると、ID トークンと userInfo はユーザー固有の ID と追加のスコープで定義された属性を返します。
- phone スコープは phone_number クレームおよび phone_number_verified クレームへのアクセスを付与します。このスコープは openid スコープを使用する場合にのみリクエストできます。
- email スコープは email クレームおよび email_verified クレームへのアクセスを付与します。このスコープは openid スコープを使用する場合にのみリクエストできます。
- aws.cognito.signin.user.admin スコープは、[UpdateUserAttributes](#) や [VerifyUserAttribute](#) など、アクセストークンを必要とする [Amazon Cognito ユーザープール API オペレーション](#) へのアクセスを許可します。
- profile スコープはクライアントが読み取り可能なすべてのユーザー属性へのアクセスを付与します。このスコープは openid スコープを使用する場合にのみリクエストできます。

スコープの詳細については、[標準 OIDC スコープ](#) のリストを参照してください。

OAuth グラントタイプ

OAuth 付与は、ユーザープールのトークンを取得する認証方法です。Amazon Cognito は、以下の付与タイプをサポートしています。これらの OAuth 付与をアプリに統合するには、ユーザープールにドメインを追加する必要があります。

認可コード付与

認可コード付与は、アプリが[トークンエンドポイント](#)を使用してユーザープールのトークンと交換できるコードを生成します。認可コードを交換すると、アプリは ID、アクセス、更新の各トークンを受け取ります。この OAuth フローは、暗黙的な付与と同様に、ユーザーのブラウザで発生します。認可コード付与は、トークンがユーザーのセッションで表示されないため、Amazon Cognito が提供する最も安全な付与です。代わりに、アプリはトークンを返すリクエストを生成し、保護されたストレージにトークンをキャッシュできます。詳細については、「[IETF RFC 6749 #1.3.1](#)」の「Authorization Code」を参照してください。

Note

パブリッククライアントアプリにおけるセキュリティのベストプラクティスとして、認可コード付与の OAuth フローのみを有効にし、Proof Key for Code Exchange (PKCE) を実装してトークン交換を制限します。PKCE では、クライアントは、元の認証リクエストで提示したのと同じシークレットをトークンエンドポイントに提供した場合にのみ、認可コードを交換できます。PKCE の詳細については、[IETF RFC 7636](#) を参照してください。

暗黙的な付与

暗黙的な付与は、[認可エンドポイント](#) からアクセストークンと ID トークンをユーザのブラウザセッションに直接配信します (更新トークンは配信しません)。暗黙的な付与では、トークンエンドポイントへの個別のリクエストは不要になりますが、PKCE とは互換性がなく、更新トークンは返されません。この付与は、認可コード付与を完了できないテストシナリオやアプリアーキテクチャに対応できます。詳細については、[IETF RFC 6749 #1.3.2](#) の「暗黙的な付与」を参照してください。認可コード付与と暗黙的な付与の両方をアプリクライアントで有効にして、必要に応じて使い分けることができます。

クライアント認証情報の付与

クライアント認証情報の付与は、マシンツーマシン (M2M) の通信に使用します。認可コード付与と暗黙的な付与は、認証された人間のユーザーにトークンを発行します。クライアント認証情報の付与は、非対話型システムから API にスコープベースの認可を付与します。アプリは、トークンエンドポイントに対してクライアント認証情報を直接リクエストし、アクセストークンを受け取ることができます。詳細については、[IETF RFC 6749 #1.3.4](#) の「クライアント認証情報」を参照してください。クライアント認証情報の付与は、クライアントシークレットを持ち、認可コード付与や暗黙的な付与をサポートしていないアプリクライアントでのみ有効化できます。

Note

ユーザーとしてクライアント認証情報フローを呼び出すことはしないため、このフローでアクセストークンに追加できるのはカスタムスコープのみです。カスタムスコープは、各自のリソースサーバー用に定義したものです。openid や profile などのデフォルトのスコープは、人間以外のユーザーには適用されません。

ID トークンはユーザー属性の検証であるため、M2M 通信には関係なく、クライアント認証情報の付与では ID トークンを発行しません。「[スコープ、M2M、リソースサーバー](#)」を参照してください。

クライアント認証情報付与は、AWS 請求書にコストを追加します。詳細については、「[Amazon Cognito の料金](#)」を参照してください。

アプリクライアントの作成

AWS マネジメントコンソール

アプリクライアントを作成する (コンソール)

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択、または[ユーザープールを作成](#)します。どちらのオプションでも、アプリケーション固有の設定でアプリケーションクライアントを設定するように求められます。
4. アプリケーションのアーキテクチャを反映するアプリケーションタイプを選択します。
5. わかりやすい識別子を使用してアプリケーションに名前を付けます。
6. [リターン URL] に入力します。
7. [アプリケーションクライアントを作成] を選択します。詳細オプションは、アプリケーションクライアントの作成後に変更できます。
8. Amazon Cognito は、アプリケーションクライアントの詳細に戻ります。アプリケーションのサンプルコードにアクセスするには、[Quick Setup ガイド] タブからプラットフォームを選択します。

AWS CLI

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

Note

コールバック URL とログアウト URL は、JSON 形式を使用することで、CLI でリモートパラメータファイルとして処理されないようにします。

```
--callback-urls ["https://example.com"]  
--logout-urls ["https://example.com"]
```

詳細については、AWS CLI コマンドリファレンスを参照してください: [create-user-pool-client](#)

Amazon Cognito user pools API

[CreateUserPoolClient](#) API リクエストを生成します。デフォルト値に設定しないすべてのパラメータには、値を指定する必要があります。

ユーザープールアプリケーションクライアント (AWS CLI および AWS API) の更新で AWS CLI、次のコマンドを入力します。

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

コマンドが成功すると、は確認 AWS CLI を返します。

```
{  
  "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
}
```

詳細については、AWS CLI コマンドリファレンス[update-user-pool-client](#)」を参照してください。

AWS API: [UpdateUserPoolClient](#)

ユーザープールアプリケーションクライアント (AWS CLI および AWS API) に関する情報の取得

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

詳細については、AWS CLI コマンドリファレンス[describe-user-pool-client](#)」を参照してください。

AWS API: [DescribeUserPoolClient](#)

ユーザープール (AWS CLI および AWS API) 内のすべてのアプリケーションクライアント情報を一覧表示する

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

詳細については、AWS CLI コマンドリファレンス[list-user-pool-clients](#)」を参照してください。

AWS API: [ListUserPoolClients](#)

ユーザープールアプリケーションクライアント (AWS CLI および AWS API) の削除

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

詳細については、AWS CLI コマンドリファレンスを参照してください: [delete-user-pool-client](#)

AWS API: [DeleteUserPoolClient](#)

ユーザープール内のユーザーデバイスの使用

Amazon Cognito ユーザープール API を使用してローカルユーザープールのユーザーをサインインさせる場合、[脅威保護](#)からのユーザーのアクティビティログをユーザーの各デバイスに関連付けることができます。また、オプションとして、ユーザーが信頼されたデバイスを使用している場合は、多要素認証 (MFA) をスキップするのを許可できます。Amazon Cognito には、デバイス情報がまだ含まれていないサインインへのレスポンスにデバイスキーが含まれます。デバイスキーの形式は **Region_UUID** です。デバイスキー、Secure Remote Password (SRP) ライブラリ、およびデバイス認証を許可するユーザープールがあれば、アプリ内のユーザーに現在のデバイスを信頼するように求めることができ、サインイン時に MFA コードの入力を求める必要がなくなります。

トピック

- [記憶済みデバイスのセットアップ](#)
- [デバイスキーの取得](#)
- [デバイスでのサインイン](#)
- [デバイスの表示、更新、削除](#)

記憶済みデバイスのセットアップ

Amazon Cognito ユーザープールでは、ユーザーの各デバイスを固有のデバイス識別子、つまりデバイスキーに関連付けることができます。サインイン時にデバイスキーを提示してデバイス認証を実行すると、信頼されたデバイスの認証フローを使用してアプリケーションを設定できます。このフローでは、アプリケーションのセキュリティ要件またはユーザーの設定に応じて、後まで MFA を使わずにサインインする選択肢をユーザーに提示できます。この期間の終了時に、アプリケーションはデバイスのステータスを [記憶されていない] に変更し、ユーザーはデバイスを記憶することを確定するまで MFA を使用してサインインする必要があります。例えば、アプリケーションは 30、60、また

は 90 日間、デバイスを信頼するようにユーザーに求める場合があります。この日付をカスタム属性に保存し、その日にデバイスの記憶されたステータスを変更できます。次に、MFA コードを送信するとともに、認証が完了した後にデバイスを再度記憶するように設定するように、ユーザーに再度求める必要があります。

1. 記憶されているデバイスは、MFA がアクティブなユーザープールでのみ MFA をオーバーライドできます。

ユーザーが記憶されているデバイスでログインする場合、認証フロー中に追加のデバイス認証を実行する必要があります。詳細については、「[デバイスでのサインイン](#)」を参照してください。

ユーザープールの [サインイン] メニューの [デバイス追跡] で、デバイスを記憶するようにユーザープールを設定します。Amazon Cognito コンソールを使用して記憶済みデバイス機能をセットアップするときは、[常に]、[User Opt-In] (ユーザーオプトイン)、および [いいえ] の 3 つのオプションがあります。

記憶しない

ユーザープールでは、ログイン時にデバイスを記憶するように求めるメッセージは表示されません。

常に記憶する

アプリがユーザーのデバイスを確認すると、ユーザープールは常にデバイスを記憶し、今後デバイスへのログインが成功しても MFA チャレンジを返しません。

ユーザーオプトイン

アプリがユーザーのデバイスを確認しても、ユーザープールは MFA チャレンジを自動的に抑制しません。デバイスを記憶するかどうかをユーザーが選択するようにユーザーに求める必要があります。

[常に記憶する] または [ユーザーオプトイン] を選択すると、ユーザーが未確認のデバイスからサインインするたびに、Amazon Cognito はデバイス ID キーとシークレットを生成します。デバイスキーは、ユーザーがデバイス認証を実行したときにアプリがユーザープールに送信する最初の識別子です。

確認済みの各ユーザーデバイスでは、自動的に記憶されるかオプトインされたかに関係なく、ユーザーがサインインするたびにデバイス識別子キーとシークレットを使用してデバイスを認証できます。

[CreateUserPool](#) または [UpdateUserPool](#) API リクエストで、ユーザープールの記憶デバイス設定を構成することもできます。詳細については、[DeviceConfiguration](#) のプロパティを参照してください。

Amazon Cognito ユーザープール API には記憶されているデバイスに対してさらに 2 つのオペレーションがあります。

1. [ListDevices](#) と [AdminListDevices](#) は、ユーザーのデバイスキーとそのメタデータのリストを返します。
2. [GetDevice](#) と [AdminGetDevice](#) は、単一のデバイスのデバイスキーとメタデータを返します。
3. [UpdateDeviceStatus](#) と [AdminUpdateDeviceStatus](#) は、ユーザーのデバイスを記憶済みまたは未記憶として設定します。
4. [ForgetDevice](#) と [AdminForgetDevice](#) は、ユーザーの確認済みデバイスをプロファイルから削除します。

名前が Admin で始まる API オペレーションはサーバー側アプリで使用するもので、IAM 認証情報を使用して認証する必要があります。詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

デバイスキーの取得

ユーザーがユーザープール API を使用してサインインし、認証パラメータに DEVICE_KEY としてデバイスキーを含めない場合、Amazon Cognito はレスポンスに新しいデバイスキーを返します。クライアント側パブリックアプリでは、デバイスキーをアプリストレージに配置することで、今後のリクエストに含められるようになります。機密のサーバー側アプリで、ブラウザー Cookie または別のクライアント側トークンをユーザーのデバイスキーで設定します。

ユーザーが信頼できるデバイスを使用してサインインする前に、アプリはデバイスキーを確認して追加情報を提供する必要があります。デバイスキー、わかりやすい名前、パスワード検証ツール、およびソルトを使用してユーザーのデバイスを確認する [ConfirmDevice](#) リクエストを Amazon Cognito に生成します。ユーザープールでオプトインデバイス認証を設定した場合、Amazon Cognito は ConfirmDevice リクエストに、ユーザーは現在のデバイスを記憶するかどうかを選択する必要があります。プロンプトで応答する必要があります。[UpdateDeviceStatus](#) リクエストでユーザーが選択した内容を返信します。

ユーザーのデバイスを確認したが、記憶されるように設定しなかった場合、Amazon Cognito は関連付けを保存しますが、デバイスキーを入力するとデバイス以外のサインインで続行します。デバイス

は、ユーザーのセキュリティとトラブルシューティングに役立つログを生成できます。確認済みで記憶されていないデバイスは、サインイン機能を利用しませんが、セキュリティモニタリングログ機能は利用します。アプリケーションクライアントの脅威保護を有効にし、デバイスのフィンガープリントをリクエストにエンコードすると、Amazon Cognito はユーザーイベントを確認済みのデバイスに関連付けます。

新しいデバイスキーを取得するには

1. [InitiateAuth](#) API リクエストを使用してユーザーのサインインセッションを開始します。
2. ユーザーのサインインセッションが完了したことをマークする JSON Web トークン (JWT) を受け取るまで、すべての認証チャレンジに [RespondToAuthChallenge](#) で応答します。
3. アプリで、Amazon Cognito がその RespondToAuthChallenge または InitiateAuth レスポンスの NewDeviceMetadata に返す値 (DeviceGroupKey および DeviceKey) を記録します。
4. ユーザー用の新しい SRP シークレット、つまりソルトおよびパスワード検証ツールを生成します。この関数は SRP ライブラリを提供する SDK で使用できます。
5. ユーザーにデバイス名の入力を求めるか、ユーザーのデバイス特性からデバイス名を生成します。
6. [ConfirmDevice](#) API リクエストでユーザーのアクセストークン、デバイスキー、デバイス名、SRP シークレットを指定します。ユーザープールがデバイスを [常に記憶する] に設定されていれば、ユーザー登録は完了です。
7. Amazon Cognito が "UserConfirmationNecessary": true で ConfirmDevice に応答した場合は、デバイスを記憶するかどうかを選択するようにユーザーに促します。ユーザーがデバイスを記憶すると断言したら、ユーザーのアクセストークン、デバイスキー、および "DeviceRememberedStatus": "remembered" を使用して [UpdateDeviceStatus](#) API リクエストを生成します。
8. Amazon Cognito にデバイスを記憶するように指示した場合、次回のサインイン時に MFA チャレンジの代わりに DEVICE_SRP_AUTH チャレンジが表示されます。

デバイスでのサインイン

ユーザーのデバイスを記憶するように設定すると、Amazon Cognito ではユーザーが同じデバイスキーでサインインするときに MFA コードを送信する必要がなくなります。デバイス認証は、MFA 認証チャレンジをデバイス認証チャレンジに置き換えるだけです。グループとしてサインインすることはできません。ユーザーはまずパスワードまたはカスタムチャレンジを使用して認証を完了する必要があります。記憶しているデバイスでのユーザーの認証プロセスは次のとおりです。

[カスタム認証チャレンジ Lambda トリガー](#)を使用するフローでデバイス認証を実行するには、[InitiateAuth](#) API リクエストで `DEVICE_KEY` パラメータを渡します。ユーザーがすべてのチャレンジに成功し、`CUSTOM_CHALLENGE` チャレンジが `true` の `issueTokens` 値を返すと、Amazon Cognito は最後の 1 つの `DEVICE_SRP_AUTH` チャレンジを返します。

デバイスでのサインインするには

1. ユーザーのデバイスキーをクライアントストレージから取得します。
2. [InitiateAuth](#) API リクエストを使用してユーザーのサインインセッションを開始します。 `USER_SRP_AUTH`、`REFRESH_TOKEN_AUTH`、`USER_PASSWORD_AUTH`、または `CUSTOM_AUTH` の `AuthFlow` を選択します。 `AuthParameters` で、ユーザーのデバイスキーを `DEVICE_KEY` パラメータに追加し、選択したログインフローに必要なその他のパラメータを含めます。
 - a. 認証チャレンジへの `PASSWORD_VERIFIER` レスポンスのパラメータで `DEVICE_KEY` を渡すこともできます。
3. レスポンスに `DEVICE_SRP_AUTH` チャレンジを受け取るまで、チャレンジレスポンスを完了します。
4. [RespondToAuthChallenge](#) API リクエストで、`DEVICE_SRP_AUTH` の `ChallengeName` と `USERNAME`、`DEVICE_KEY`、および `SRP_A` のパラメータを送信します。
5. Amazon Cognito は `DEVICE_PASSWORD_VERIFIER` チャレンジで応答します。このチャレンジレスポンスには、`SECRET_BLOCK` および `SRP_B` の値が含まれます。
6. SRP ライブラリを使用して `PASSWORD_CLAIM_SIGNATURE`、`PASSWORD_CLAIM_SECRET_BLOCK`、`TIMESTAMP`、`USERNAME`、`DEVICE_KEY` パラメータを生成して送信します。これらは追加の `RespondToAuthChallenge` リクエストで送信してください。
7. ユーザーの JWT を受け取るまで、追加のチャレンジを完了してください。

次の擬似コードは、`DEVICE_PASSWORD_VERIFIER` チャレンジレスポンスの値を計算する方法を示しています。デバイスによる SRP 認証の場合は、ユーザーの新しい SRP シークレット (新しい高エントロピーパスワード `DeviceSecret`、ソルト、および関連するパスワード検証子) を生成します。これらの値は、ユーザーの SRP 認証に使用されるパスワード、ソルト、検証子とは異なります。これらはデバイス認証専用であり、デバイスにのみ保存されます。ユーザーのデバイスの SRP シークレットを生成する関数は、[SRP ライブラリ](#) にあり、さまざまな SDK で利用できます。

```
PASSWORD_CLAIM_SECRET_BLOCK = SECRET_BLOCK
TIMESTAMP = "Tue May 7 00:09:40 UTC 2025"
```



```
k = SHA256(N || g) as a non-negative integer in big-endian
u = SHA256(SRP_A || SRP_B) as a non-negative integer in big-endian
x = SHA256(salt || SHA256(DeviceGroupKey || DeviceKey || ":" || DeviceSecret)) as a
  non-negative integer in big-endian
S_USER = (SRP_B - k * g^x)^(a + u * x) % N
K_USER = HKDF_HMAC_SHA256(salt=u, ikm=S_USER, info="Caldera Derived Key", length=16
  bytes)
PASSWORD_CLAIM_SIGNATURE = Base64(HMAC_SHA256(key=K_USER, message=(DeviceGroupKey ||
  DeviceKey || PASSWORD_CLAIM_SECRET_BLOCK || TIMESTAMP)))
```

デバイスの表示、更新、削除

Amazon Cognito API を使用して、アプリに次の機能を実装できます。

1. ユーザーの現在のデバイスに関する情報を表示します。
2. ユーザーのすべてのデバイスのリストを表示します。
3. デバイスの「記憶済み」状態を解除する
4. デバイスの記憶状態を更新します。

以下の説明の API リクエストを承認するアクセストークンに

は、`aws.cognito.signin.user.admin` スコープを含める必要があります。Amazon Cognito は、Amazon Cognito ユーザープール API を使用して生成するすべてのアクセストークンに、このスコープのクレームを追加します。サードパーティ IdP は、Amazon Cognito を認証するユーザーのデバイスと MFA を個別に管理する必要があります。マネージドログインでは、`aws.cognito.signin.user.admin` スコープをリクエストできますが、マネージドログインではデバイス情報が高度なセキュリティのユーザーログに自動的に追加され、デバイスが記憶されることはありません。

デバイスに関する情報を表示する

ユーザーのデバイスに関する情報をクエリして、そのデバイスが現在も使用されているかどうかを判断できます。例えば、記憶されているデバイスが 90 日間サインインしていない場合に、そのデバイスを非アクティブ化することができます。

- パブリッククライアントアプリにユーザーのデバイス情報を表示するには、ユーザーのアクセスキーとデバイスキーを [GetDevice](#) API リクエストで送信します。
- 機密クライアントアプリにユーザーのデバイス情報を表示するには、AWS 認証情報を使用して [AdminGetDevice](#) API リクエストに署名し、ユーザーのユーザー名、デバイスキー、ユーザープールを送信します。

ユーザーのすべてのデバイスのリストを表示する

ユーザーのすべてのデバイスとそのプロパティのリストを表示できます。例えば、現在のデバイスが記憶されているデバイスと一致することを確認することが可能です。

- パブリッククライアントアプリでは、[ListDevices](#) API リクエストでユーザーのアクセストークンを送信します。
- 機密クライアントアプリで、AWS 認証情報を使用して [AdminListDevices](#) API リクエストに署名し、ユーザーのユーザー名とユーザープールを送信します。

デバイスの「記憶済み」状態を解除する

ユーザーのデバイスキーは削除できます。これは、ユーザーがデバイスを使用しなくなったと判断した場合や、異常なアクティビティを検出してユーザーに MFA を再度完了するように促す場合に役立ちます。デバイスを後で再登録するには、新しいデバイスキーを生成して保存する必要があります。

- パブリッククライアントアプリでは、[ForgetDevice](#) API リクエストでユーザーのデバイスキーとアクセストークンを送信します。
- 機密クライアントアプリでは、[AdminForgetDevice](#) API リクエストでユーザーのデバイスキーとアクセストークンを送信します。

ユーザープール分析に Amazon Pinpoint を使用する

Note

サポート終了通知: 2026 年 10 月 30 日に、AWS は Amazon Pinpoint のサポートを終了します。2026 年 10 月 30 日を過ぎると、Amazon Pinpoint コンソールまたは Amazon Pinpoint のリソース (エンドポイント、セグメント、キャンペーン、ジャーニー、分析) にアクセスできなくなります。詳細については、「[Amazon Pinpoint のサポート終了](#)」を参照してください。注: SMS、音声、モバイルプッシュ、OTP、電話番号の検証に関連する APIs は、この変更の影響を受けず、AWS エンドユーザーメッセージングでサポートされています。

Amazon Cognito ユーザープールは、Amazon Cognito ユーザープールの分析を提供し、Amazon Pinpoint キャンペーンの利用者データを強化するために Amazon Pinpoint と統合されています。Amazon Pinpoint は、プッシュ通知を使用するモバイルアプリケーションでのユーザーエンゲージメントを促進するために、分析と目的を絞ったキャンペーンを提供します。Amazon Cognito

ユーザープールでの Amazon Pinpoint 分析のサポートを使用すると、Amazon Pinpoint コンソールでユーザープールへのサインアップ、サインイン、失敗した認証、デイリーアクティブユーザー数 (DAU)、および月間アクティブユーザー数 (MAU) を追跡できます。デバイスプラットフォーム、デバイスロケール、およびアプリケーションのバージョンなどのデータを異なる日付範囲や属性で表示できます。

また、アプリケーションにカスタム属性を設定することもできます。これらは、Amazon Pinpoint でユーザーを分割し、的を絞ったプッシュ通知をユーザーに送信するために使用できます。Amazon Cognito コンソールの [アプリケーションクライアント] メニューで、アプリケーションクライアントの [分析] 設定の [Amazon Pinpoint とユーザー属性データを共有] を選択すると、Amazon Pinpoint は、ユーザーの E メールアドレスと電話番号用に追加のエンドポイントを作成します。

Amazon Cognito コンソールを使用してユーザープールの Amazon Pinpoint 分析をアクティブ化すると、Amazon Cognito がユーザープールの Amazon Pinpoint に API リクエストを送信するときに引き継ぐ[サービスにリンクされたロール](#)も作成されます。分析設定を追加する IAM プリンシパルには、[CreateServiceLinkedRole](#) 権限が必要です。サービスにリンクされたロールは、[AWSServiceRoleForAmazonCognitoIdp](#) です。詳細については、「[Amazon Cognito のサービスリンクロールの使用](#)」を参照してください。

Amazon Cognito API で `AnalyticsConfiguration` をアプリケーションクライアントに適用する際、Amazon Pinpoint にカスタム IAM ロールを割り当てて、そのロールを引き継ぐ外部 ID を割り当てることができます。ロールは `cognito-idp` サービスプリンシパルを信頼している必要があり、ロール信頼ポリシーで外部 ID が必要な場合は、`AnalyticsConfiguration` と一致する必要があります。Amazon Pinpoint プロジェクトには、ロール `cognito-idp:Describe*` 権限と以下の権限を付与する必要があります。

- `mobiletargeting:UpdateEndpoint`
- `mobiletargeting:PutEvents`

Amazon Cognito と Amazon Pinpoint の利用可能なリージョン

次の表は、次のいずれかの条件を満たす Amazon Cognito と Amazon Pinpoint 間の AWS リージョンマッピングを示しています。

- Amazon Pinpoint プロジェクトは、米国東部 (バージニア北部) (us-east-1) リージョンでのみ使用できます。
- Amazon Pinpoint プロジェクトは、同じリージョンまたは米国東部 (バージニア北部) (us-east-1) リージョンで使用できます。

デフォルトでは、Amazon Cognito は同じ AWS リージョンの Amazon Pinpoint プロジェクトにのみ分析を送信できます。このルールの例外は、次の表のリージョンと Amazon Pinpoint が利用できないリージョンです。

Amazon Pinpoint は、次のリージョンでは利用できません。これらのリージョンの Amazon Cognito ユーザープールは分析をサポートしていません。

- 欧州 (ミラノ)
- 中東 (バーレーン)
- アジアパシフィック (大阪)
- イスラエル (テルアビブ)
- アフリカ (ケープタウン)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (マレーシア)

この表は、Amazon Cognito ユーザープールを構築したリージョンと、Amazon Pinpoint の対応するリージョンとの関係を示しています。Amazon Pinpoint プロジェクトを Amazon Cognito と統合するには、使用可能なリージョンに設定する必要があります。

Amazon Cognito ユーザープールのリージョン	Amazon Pinpoint プロジェクトのリージョン
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

Amazon Cognito ユーザープールのリージョン	Amazon Pinpoint プロジェクトのリージョン
us-east-1	us-east-1
us-east-2	us-east-1
us-west-2	us-east-1、us-west-2

リージョンマッピングの例

- ap-northeast-1 でユーザープールを作成する場合、Amazon Pinpoint プロジェクトは us-east-1 で作成する必要があります。
- ap-south-1 でユーザープールを作成する場合、Amazon Pinpoint プロジェクトは us-east-1 または ap-south-1 のいずれかで作成できます。

Note

上記の表 AWS リージョン を除くすべての について、Amazon Cognito はユーザープールと同じリージョンの Amazon Pinpoint プロジェクトのみを使用できます。ユーザープールを構築したリージョンで Amazon Pinpoint が利用できず、表にも記載されていない場合、Amazon Cognito はそのリージョンで Amazon Pinpoint 分析をサポートしません。詳細な AWS リージョン 情報については、「[Amazon Pinpoint エンドポイントとクォータ](#)」を参照してください。

Amazon Pinpoint 分析設定の指定 (AWS マネジメントコンソール)

Amazon Cognito ユーザープールを設定して、分析データを Amazon Pinpoint に送信できます。Amazon Cognito は、ローカルユーザーの分析データのみを Amazon Pinpoint に送信します。ユーザープールを Amazon Pinpoint プロジェクトに関連付けるように設定したら、API リクエストに AnalyticsMetadata を含める必要があります。詳細については、「[アプリを Amazon Pinpoint と統合する](#)」を参照してください。

分析の設定を指定する

1. [Amazon Cognito コンソール](#)に移動します。AWS 認証情報を求められる場合があります。
2. [User Pools] (ユーザープール) を選択して、リストから既存のユーザープールを選択します。

3. [アプリケーションクライアント] メニューを選択し、更新するアプリケーションクライアントを選択します。
4. [ピンポイント分析] の下で [分析] タブの [有効化] を選択します。
5. [Pinpoint Region] (Pinpoint リージョン) を選択します。
6. [Amazon Pinpoint project] (Amazon Pinpoint プロジェクト) を選択するか、[Create Amazon Pinpoint project] (Amazon Pinpoint プロジェクトを作成する) を選択します。

 Note

Amazon Pinpoint プロジェクト ID は、Amazon Pinpoint プロジェクトに固有の 32 文字の文字列です。これは Amazon Pinpoint コンソールにリストされています。

複数の Amazon Cognito アプリケーションを単一の Amazon Pinpoint プロジェクトにマッピングできますが、各 Amazon Cognito アプリは 1 つの Amazon Pinpoint プロジェクトにしかマップできません。

Amazon Pinpoint では、各プロジェクトを単一のアプリにする必要があります。例えば、ゲームデベロッパーが 2 つのゲームを持っている場合、両方のゲームが同じ Amazon Cognito ユーザープールを使用しているとしても、各ゲームを個別の Amazon Pinpoint プロジェクトにする必要があります。Amazon Pinpoint プロジェクトの詳細については、「[Amazon Pinpoint でプロジェクトを作成する](#)」を参照してください。

7. Amazon Cognito が E メールアドレスと電話番号を Amazon Pinpoint に送信し、ユーザー用の追加のエンドポイントを作成する場合は、[User data sharing] (ユーザーデータの共有) で [Share user data with Amazon Pinpoint] (Amazon Pinpoint でユーザーデータを共有する) を選択します。ユーザーが E メールアドレスと電話番号を確認した後、Amazon Cognito はそれらがユーザーアカウントで利用可能な場合にのみ Amazon Pinpoint と共有します。

 Note

エンドポイントは、Amazon Pinpoint でプッシュ通知を送信することができるユーザーデバイスを一意に識別します。エンドポイントの詳細については、Amazon Pinpoint デベロッパーガイドの「[エンドポイントの追加](#)」を参照してください。

8. [Save changes] (変更の保存) をクリックします。

Amazon Pinpoint 分析設定の指定 (AWS CLI および AWS API)

以下のコマンドを使用して、ユーザープールの Amazon Pinpoint 分析設定を指定します。

アプリケーション作成時にユーザープールの既存のクライアントアプリケーションの分析設定を指定する

- AWS CLI: `aws cognito-idp create-user-pool-client`
- AWS API: [CreateUserPoolClient](#)

ユーザープールの既存のクライアントアプリケーションの分析設定を更新する

- AWS CLI: `aws cognito-idp update-user-pool-client`
- AWS API: [UpdateUserPoolClient](#)

Note

Amazon Cognito は、ApplicationArn の使用時にリージョン内の統合をサポートします。

アプリを Amazon Pinpoint と統合する

ユーザープール API の Amazon Cognito ローカルユーザーの分析メタデータを Amazon Pinpoint に公開できます。

ローカルユーザー

サードパーティー ID プロバイダー (IdP) を通じてサインインするのではなく、アカウントにサインアップしたユーザー、またはユーザープールで作成されたユーザー。

ユーザープール API

カスタムユーザーインターフェイス (UI) でアプリケーションを使用して AWS SDK と統合できるオペレーション。マネージドログインでサインインしたフェデレーションユーザーまたはローカルユーザーの分析メタデータを渡すことはできません。ユーザープール API オペレーションのリストについては、「[Amazon Cognito API リファレンス](#)」を参照してください。

キャンペーンに公開するようにユーザープールを設定すると、Amazon Cognito は次の API オペレーションのためのメタデータを Amazon Pinpoint に渡します。

- AdminInitiateAuth
- AdminRespondToAuthChallenge

- ConfirmForgotPassword
- ConfirmSignUp
- ForgotPassword
- InitiateAuth
- ResendConfirmationCode
- RespondToAuthChallenge
- SignUp

ユーザーのセッションに関するメタデータを Amazon Pinpoint キャンペーンに渡すには、API リクエストの AnalyticsMetadata パラメータに AnalyticsEndpointId 値を含めます。JavaScript の例については、「AWS ナレッジセンター」の「[Amazon Pinpoint ダッシュボードに Amazon Cognito ユーザープールの分析が表示されないのはなぜですか?](#)」を参照してください。

Amazon Cognito ユーザープールの E メール設定

アプリケーションでの特定のイベントにより、Amazon Cognito がユーザーに E メールを送信する場合があります。例えば、Eメールの検証を必須とするようにユーザープールを設定している場合、ユーザーがアプリの新しいアカウントにサインアップする、またはパスワードをリセットする場合に Amazon Cognito が E メールを送信します。E メールを開始するアクションに応じて、検証コードまたは仮パスワードが含まれます。

E メール配信を処理するには、次のいずれかのオプションを使用できます。

- Amazon Cognito サービスに組み込まれている[デフォルトの E メール設定](#)。
- [Amazon Simple Email Service \(Amazon SES\) の設定](#)。

ユーザープールを作成した後に、配信オプションを変更できます。

Amazon Cognito は、ユーザーが入力できるコードか選択可能な URL リンクのいずれかを含む E メールメッセージをユーザーに送信します。次の表は、E メールメッセージを生成できるイベントを示しています。

メッセージオプション

アクティビティ	API オペレーション:	配信オプション	形式オプション	カスタマイズ可能	メッセージテンプレート
Forgot password	ForgotPassword , AdminResetUserPassword	Email, SMS	code	Yes	検証メッセージ
Invitation	AdminCreateUser	Email, SMS	code	Yes	招待メッセージ
Self-registration	SignUp , ResendConfirmationCode	Email, SMS	code, link	Yes	検証メッセージ
Email address or phone number verification	UpdateUserAttributes , AdminUpdateUserAttributes , GetUserAttributeVerificationCode	Email, SMS	code	Yes	検証メッセージ
Multi-factor authentication (MFA)	AdminInitiateAuth , InitiateAuth	Email ¹ , SMS, authenticator app	code	Yes ²	MFA メッセージ
One-time password authentication (OTP)	AdminInitiateAuth , InitiateAuth	Email ¹ , SMS	code	Yes	MFA メッセージ ³

¹ エッセンシャル以上の[機能プラン](#)と [Amazon SES E メール設定](#)が必要です。

² SMS メッセージおよび E メールメッセージの場合。

³ ユーザープールで MFA が必須またはオプションである場合にのみ、MFA メッセージテンプレートをカスタマイズできます。MFA が非アクティブの場合、Amazon Cognito はデフォルトのテンプレートを使用してワンタイムパスワードを送信します。

Amazon SES はメールメッセージに対して料金を請求します。詳細については、「[Amazon SES の料金](#)」を参照してください。

E メール MFA の詳細については、「[SMS メッセージ MFA と E メールメッセージ MFA](#)」を参照してください。

Amazon Cognito は、1 つの送信先に対する短期間での E メールや SMS メッセージの追加配信を防止する場合があります。ユーザープールが影響を受けていると思われる場合は、[メッセージ配信エラーのログ](#)を設定および確認してから、アカウントチームにお問い合わせください。

デフォルトの E メール設定

Amazon Cognito は、デフォルトの E メール設定を使用して E メール配信を処理できます。デフォルトのオプションを使用すると、Amazon Cognito は、ユーザープールに毎日送信するメールの数を制限します。サービスの制限に関する詳細については、「[Amazon Cognito のクォータ](#)」を参照してください。一般的な実稼働環境では、デフォルトの E メール制限は必要な配信ボリュームを下回っています。大規模なデリバリーボリュームを有効にするには、Amazon SES E メール設定を使用する必要があります。

デフォルトの設定を使用する場合は、AWS によって管理される Amazon SES リソースを使用して E メールメッセージを送信します。Amazon SES は、[ハードバウンス](#)を返す E メールアドレスを[アカウントレベルのサプレッションリスト](#)または[グローバルサプレッションリスト](#)に追加します。配信不能な E メールアドレスが後で配信可能になった場合、ユーザープールがデフォルト設定を使用するように設定されている間は、サプレッションリストからの削除を制御できません。E メールアドレスは AWS、マネージドサプレッションリストに無期限に残すことができます。配信不能な E メールアドレスを管理するには、次のセクションで説明するように、アカウントレベルのサプレッションリストで Amazon SES E メール設定を使用します。

デフォルトの E メール設定を使用すると、オプションでは、次のいずれかの E メールアドレスを FROM アドレスとして使用できます。

- デフォルトの E メールアドレス (no-reply@verificationemail.com)。
- カスタム E メールアドレス。独自の E メールアドレスを使用する前に、それを Amazon SES で検証し、このアドレスを使用するための許可を Amazon Cognito に付与する必要があります。

Amazon SES の E メール設定

アプリケーションは、デフォルトのオプションで利用できるよりも大きいボリュームを必要とする可能性があります。可能なデリバリーボリュームを増やすには、Amazon SES リソースとユーザープールを使用してユーザーに E メールを送信します。独自の Amazon SES 設定で E メールメッセージを送信するときに、[E メール送信アクティビティをモニタリングする](#)こともできます。

Amazon SES の設定を使用する前に、1 つ、または複数の E メールアドレスまたはドメインを Amazon SES で検証する必要があります。検証済み E メールアドレスまたは検証済みドメインのアドレスを、ユーザープールに割り当てる FROM E メールアドレスとして使用します。Amazon Cognito が E メールをユーザーに送信するときは、Amazon SES を呼び出し、その E メールアドレスを使用するようになります。

Amazon SES の設定を使用する場合は、次の条件が適用されます。

- ユーザープールの E メール配信制限が、AWS アカウントの Amazon SES 検証済み E メールアドレスに適用される制限と同じになります。
- Amazon SES のアカウントレベルのサブレーションリストで、[グローバルサブレーションリスト](#)を上書きして、配信不能な E メールアドレスへのメッセージを管理することができます。アカウントレベルのサブレーションリストを使用すると、E メールメッセージのバウンスが送信者としてのアカウントの評価に影響します。詳細については、Amazon Simple Email Service デベロッパーガイドの「[Amazon SES アカウントレベルのサブレーションリストの使用](#)」を参照してください。

Amazon SES の E メール設定リージョン

ユーザープール AWS リージョンを作成するには、Amazon SES での E メールメッセージの設定に関する 3 つの要件のいずれかがあります。ユーザープールと同じリージョン、同じリージョンを含む複数のリージョン、または 1 つ以上のリモートリージョンの Amazon SES から E メールメッセージを送信できます。最高のパフォーマンスを得るには、ユーザープールと同じリージョンというオプションがある場合は、これで Amazon SES 検証済み ID の E メールメッセージを送信します。

Amazon SES 検証済み ID のリージョン要件のカテゴリ

リージョン内のみ

ユーザープールは、ユーザープール AWS リージョンと同じで検証済み ID を持つ E メールメッセージを送信できます。カスタム FROM E メールアドレスのないデフォルトの E メール設定で

は、Amazon Cognito は同じリージョンで no-reply@verificationemail.com 検証済み ID を使用します。

下位互換性あり

ユーザープールは、同じリージョン AWS リージョン または次のいずれかの代替リージョンで、検証済み ID を含む E メールメッセージを送信できます。

- 米国東部 (バージニア北部)
- 米国西部 (オレゴン)
- 欧州 (アイルランド)

この機能は、サービスの起動時に Amazon Cognito の要件を満たすように作成した可能性のあるユーザープールリソースの継続性をサポートします。その期間のユーザープールは、限られた数の ID が検証済みの E メールメッセージのみを送信できます AWS リージョン。カスタム FROM E メールアドレスのないデフォルトの E メール設定では、Amazon Cognito は同じリージョンで no-reply@verificationemail.com 検証済み ID を使用します。

代替リージョン

ユーザープールは、ユーザープールリージョン外の代替 で、検証 AWS リージョン 済み ID を含む E メールメッセージを送信できます。この設定は、Amazon Cognito が利用可能なリージョンで Amazon SES が利用できない場合に発生します。

代替リージョンでの検証済み ID の Amazon SES 送信認可ポリシーは、発信元のリージョンの Amazon Cognito サービスプリンシパルを信頼する必要があります。詳細については、「[デフォルトの E メール設定を使用するアクセス許可を付与するには](#)」を参照してください。

これらのリージョンの一部では、Amazon Cognito は、COGNITO_DEFAULT のデフォルト E メール設定のために E メールメッセージを 2 つの代替リージョンに分割します。このような場合、カスタム FROM E メールアドレスを使用するには、各代替リージョンでの検証済み ID の Amazon SES 送信認可ポリシーが、発信元のリージョンの Amazon Cognito サービスプリンシパルを信頼する必要があります。詳細については、「[デフォルトの E メール設定を使用するアクセス許可を付与するには](#)」を参照してください。これらのリージョンでの DEVELOPER の Amazon SES E メール設定では、最初にリストされたリージョンで検証済み ID を使用し、ユーザープールリージョンで Amazon Cognito サービスプリンシパルを信頼するように設定する必要があります。例えば、中東 (アラブ首長国連邦) のユーザープールで、cognito-idp.me-central-1.amazonaws.com を信頼するように欧州 (フランクフルト) で検証済み ID を設定します。カスタム FROM E メールアドレスのないデフォルトの E メール設定では、Amazon Cognito は各リージョンで no-reply@verificationemail.com 検証済み ID を使用します。

Note

以下の条件の組み合わせでは、リージョン要素のワイルドカードを使用して [EmailConfiguration](#) の SourceArn パラメータを `arn:${Partition}:ses:*:${Account}:identity/${IdentityName}` 形式で指定する必要があります。これにより、ユーザープールは、AWS アカウント 両方 ので同じ検証済み ID を持つ E メールメッセージを送信できます AWS リージョン。

- EmailSendingAccount は COGNITO_DEFAULT です。
- カスタム FROM アドレスを使用します。
- ユーザープールは代替リージョンで E メールを送信します。
- ユーザープールには、次の「Amazon SES でサポートされているリージョン」の表に指定されている 2 番目¹ の代替リージョンがあります。

AWS SDK、Amazon Cognito API または CLI を使用してプログラムでユーザープールを作成する場合、または AWS CloudFormation ユーザープールは AWS CDK、EmailConfiguration の SourceArn パラメータがユーザープールに指定する Amazon SES ID を含む E メールメッセージを送信します。 [EmailConfiguration](#) Amazon SES ID は、サポートされている を占有する必要があります AWS リージョン。 EmailSendingAccount が COGNITO_DEFAULT で SourceArn パラメータを指定しない場合、Amazon Cognito はユーザープールを作成したリージョン内のリソースを使用して no-reply@verificationemail.com から E メールメッセージを送信します。

次の表は、Amazon Cognito で Amazon SES ID を AWS リージョン 使用できる を示しています。
Amazon Cognito

ユーザープールリージョン	リージョンオプション	Amazon SES がサポートされるリージョン
米国東部 (バージニア北部)	下位互換性あり	米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
米国東部 (オハイオ)	下位互換性あり	米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)

ユーザープールリージョン	リージョンオプション	Amazon SES がサポートされるリージョン
米国西部 (北カリフォルニア)	リージョン内のみ	米国西部 (北カリフォルニア)
米国西部 (オレゴン)	下位互換性あり	米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
カナダ (中部)	下位互換性あり	カナダ (中部)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
カナダ西部 (カルガリー)	代替リージョン	カナダ (中部)、米国西部 (北カリフォルニア) ¹
メキシコ (中部)	代替リージョン	米国東部 (バージニア北部)、米国西部 (オレゴン) ¹
アジアパシフィック (東京)	下位互換性あり	アジアパシフィック (東京)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
アジアパシフィック (香港)	代替リージョン	アジアパシフィック (シンガポール)、アジアパシフィック (東京) ¹
アジアパシフィック (ソウル)	下位互換性あり	アジアパシフィック (ソウル)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
アジアパシフィック (マレーシア)	代替リージョン	アジアパシフィック (シドニー)、アジアパシフィック (シンガポール) ¹

ユーザープールリージョン	リージョンオプション	Amazon SES がサポートされるリージョン
アジアパシフィック (タイ)	代替リージョン	アジアパシフィック (シンガポール)、アジアパシフィック (ムンバイ) ¹
アジアパシフィック (ムンバイ)	下位互換性あり	アジアパシフィック (ムンバイ)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
アジアパシフィック (ハイデラバード)	代替リージョン	アジアパシフィック (ムンバイ)、アジアパシフィック (シンガポール) ¹
アジアパシフィック (シンガポール)	下位互換性あり	アジアパシフィック (シンガポール)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
アジアパシフィック (シドニー)	下位互換性あり	アジアパシフィック (シドニー)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
アジアパシフィック (大阪)	リージョン内のみ	アジアパシフィック (大阪)
アジアパシフィック (ジャカルタ)	リージョン内のみ	アジアパシフィック (ジャカルタ)
アジアパシフィック (メルボルン)	代替リージョン	アジアパシフィック (シドニー)、アジアパシフィック (シンガポール) ¹
欧州 (アイルランド)	下位互換性あり	米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)

ユーザープールリージョン	リージョンオプション	Amazon SES がサポートされるリージョン
欧州 (ロンドン)	下位互換性あり	欧州 (ロンドン)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
欧州 (パリ)	リージョン内のみ	欧州 (パリ)
欧州 (フランクフルト)	下位互換性あり	欧州 (フランクフルト)、米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド)
欧州 (チューリッヒ)	代替リージョン	欧州 (フランクフルト)、欧州 (ロンドン) ¹
欧州 (ストックホルム)	リージョン内のみ	欧州 (ストックホルム)
欧州 (ミラノ)	リージョン内のみ	欧州 (ミラノ)
欧州 (スペイン)	代替リージョン	欧州 (パリ)、欧州 (ストックホルム) ¹
中東 (バーレーン)	リージョン内のみ	中東 (バーレーン)
中東 (アラブ首長国連邦)	代替リージョン	欧州 (フランクフルト)、欧州 (ロンドン) ¹
南米 (サンパウロ)	リージョン内のみ	南米 (サンパウロ)
イスラエル (テルアビブ)	リージョン内のみ	イスラエル (テルアビブ)
アフリカ (ケープタウン)	リージョン内のみ	アフリカ (ケープタウン)

¹ デフォルトの E メール設定を持つユーザープールで使用されます。Amazon Cognito は、各リージョンで同じ E メールアドレスを持つ検証済み ID に

E メールメッセージを配信します。カスタム FROM アドレスを使用するには、arn:`arn:{{Partition}}:ses:*:{{Account}}:identity/{{IdentityName}}` 形式の SourceArn パラメータを使用して EmailConfiguration を設定します。

ユーザープールの E メールの設定

ユーザープールの E メール設定を指定するには、以下のステップを完了します。使用する設定に応じて、Amazon SES、AWS Identity and Access Management (IAM)、および Amazon Cognito での IAM アクセス許可が必要になる場合があります。

Note

これらのステップで作成したリソースを AWS アカウント間で共有することはできません。例えば、あるアカウントのユーザープールを設定してから、別のアカウントの Amazon SES E メールアドレスで使用することはできません。複数のアカウントで Amazon Cognito を使用する場合は、それぞれのアカウントでこれらのステップを繰り返すようにしてください。

ステップ 1: Amazon SES での E メールアドレスを検証する

次のいずれかを行う場合は、ユーザープールを設定する前に、Amazon SES で 1 つ、または複数の E メールアドレスを検証する必要があります。

- FROM アドレスとして独自の E メールアドレスを使用する
- Amazon SES の設定を使用して E メール配信を処理する

E メールアドレスを検証することにより、そのアドレスを所有していることを確認する (不正使用の防止に役立ちます)

Amazon SES での E メールアドレスの検証については、Amazon Simple Email Service デベロッパーガイドの「[E メールアドレスの検証](#)」を参照してください。Amazon SES でのドメインの検証の詳細については、「[ドメインの検証](#)」を参照してください。

ステップ 2: Amazon SES サンドボックスからアカウントを移動する

デフォルトの Amazon Cognito E メール設定を使用している場合は、このステップを省略します。

で最初に Amazon SES を使用すると AWS リージョン、そのリージョン AWS アカウント の Amazon SES サンドボックスに が配置されます。Amazon SES は、サンドボックスを使用し

て不正使用や悪用を防ぎます。Amazon SES 設定を使用して E メール配信を処理している場合は、Amazon Cognito がユーザーに E メールを送信する前に、サンドボックスから AWS アカウントを移動させる必要があります。

サンドボックスでは、Amazon SES が、送信できる E メールの数と、それらの送信先に制限を課します。E メールは、Amazon SES で検証したアドレスとドメインのみに送信できます。または、Amazon SES メールボックスシミュレーターのアドレスにも E メールを送信できます。がサンドボックスに AWS アカウント 残っている間は、本番環境のアプリケーションに Amazon SES 設定を使用しないでください。この状況では、Amazon Cognito がユーザーの E メールアドレスにメッセージを送信することができません。

サンドボックス AWS アカウント から を削除するには、[Amazon SES サンドボックスからの移動](#)」を参照してください。

ステップ 3: Amazon Cognito に E メール許可を付与する

Amazon Cognito がユーザーに E メールを送信する前に、特定の許可を Amazon Cognito に付与する必要がある場合があります。付与するアクセス許可、およびそれらを付与するために使用するプロセスは、デフォルトの E メール設定と Amazon SES の設定のどちらを使用しているかに応じて異なります。

デフォルトの E メール設定を使用するアクセス許可を付与するには

このステップは、Cognito で E メールを送信するようにユーザープールを設定するか、EmailSendingAccount を COGNITO_DEFAULT に設定する場合にのみ実施します。

デフォルトの E メール設定では、ユーザープールは次のいずれかのアドレスを使用して E メールを送信できます。

- デフォルトのアドレス no-reply@verificationemail.com。
- Amazon SES での検証された E メールアドレスまたはドメインからのカスタム FROM アドレス。

カスタムアドレスを使用している場合、Amazon Cognito には、そのアドレスからユーザーに E メールを送信するための追加のアクセス許可が必要です。これらのアクセス許可は、Amazon SES でアドレス用またはドメイン用の[送信認可ポリシー](#)によって付与されます。Amazon Cognito コンソールを使用してユーザープールにカスタムアドレスを追加する場合、ポリシーは Amazon SES 検証済み E メールアドレスに自動的にアタッチされます。ただし、AWS CLI や Amazon Cognito API など、コンソールの外部でユーザープールを設定する場合は、[Amazon SES コンソール](#)または [PutIdentityPolicy](#) API を使用してポリシーをアタッチする必要があります。

Note

FROM アドレスは、AWS CLI または Amazon Cognito API を使用して検証済みドメインでのみ設定できます。

送信承認ポリシーは、Amazon Cognito を使用して Amazon SES を呼び出すアカウントリソースに基づいてアクセスを許可、または拒否します。リソースベースのポリシーに関する詳細については、[IAM ユーザーガイド](#)を参照してください。リソースベースのポリシーの例についても、[Amazon SES デベロッパーガイド](#)で確認いただけます。

Example送信承認ポリシー

次の送信承認ポリシーの例では、Amazon SES 検証済み ID を使用するための限定された権限が Amazon Cognito に付与されています。Amazon Cognito は、条件 `aws:SourceArn` のユーザープールと条件 `aws:SourceAccount` のアカウントの両方に代わって送信する場合にのみ、E メールメッセージを送信できます。

Regions with Amazon SES

ユーザープールリージョンまたは代替リージョンの送信認可ポリシーでは、E メールメッセージを送信することを Amazon Cognito サービスプリンシパルに許可する必要があります。詳細については、[リージョン表](#)を参照してください。ユーザープールリージョンが Amazon SES リージョンで少なくとも 1 つの値と一致する場合は、次の例のグローバルサービスプリンシパルを使用して送信認可ポリシーを設定します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "stmtnt1234567891234",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "email.cognito-idp.amazonaws.com"
        ]
      },
      "Action": [
```

```

        "SES:SendEmail",
        "SES:SendRawEmail"
    ],
    "Resource": "arn:aws:ses:us-east-1:111122223333:identity/support@example.com",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:cognito-idp:us-east-1:111122223333:userpool/us-east-1_EXAMPLE"
        }
    }
}
]
}

```

Opt-in Regions without Amazon SES

Amazon SES は、Amazon Cognito AWS リージョン が利用可能なすべてのオプトインで利用できるわけではありません。中東 (アラブ首長国連邦) が例として挙げられますが、欧州 (フランクフルト) (eu-central-1) で検証済み ID を持つ E メールのみを送信できます。デフォルトの E メール設定を持つユーザープールでは、Amazon Cognito は 2 つのリージョンのそれぞれで検証済み ID を含む E メールメッセージも送信します。中東 (アラブ首長国連邦) の場合、追加のリージョンは欧州 (ロンドン) です。両方のリージョンの送信認可ポリシーを更新する必要があります。

各代替リージョンの送信認可ポリシーでは、E メールメッセージを送信することを、ユーザープールオプトインリージョンの Amazon Cognito サービスプリンシパルに許可する必要があります。詳細については、[リージョン表](#)を参照してください。リージョンが代替リージョンとしてマークされている場合は、次の例のように、リージョンサービスプリンシパルを使用して送信認可ポリシーを設定します。必要に応じて、サンプルリージョン識別子 *me-central-1* を必要なリージョン ID に置き換えます。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [

```

```

    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cognito-idp.me-central-1.amazonaws.com"
        ]
      },
      "Action": [
        "SES:SendEmail",
        "SES:SendRawEmail"
      ],
      "Resource": "arn:aws:ses:us-east-1:111122223333:identity/support@example.com",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:cognito-idp:us-east-1:111122223333:userpool/us-east-1_EXAMPLE"
        }
      }
    }
  ]
}

```

ポリシー構文の詳細については、「Amazon Simple Email Service デベロッパーガイド」の「[Amazon SES 送信承認ポリシー](#)」を参照してください。

その他の例については、「Amazon Simple Email Service デベロッパーガイド」の「[Amazon SES 送信承認ポリシーの例](#)」を参照してください。

Amazon SES の設定を使用する許可を付与する

Amazon SES の設定を使用するようにユーザープールを設定する場合、Amazon Cognito には、ユーザーに E メールを送信するときに代理で Amazon SES を呼び出すための追加の許可が必要です。この承認は、IAM サービスで付与されます。

このオプションでユーザープールを設定すると、Amazon Cognito が AWS アカウントにサービスリソースロール (IAM ロールの一種) を作成します。このロールには、Amazon Cognito が Amazon SES にアクセスし、お客様のアドレスを使用して E メールを送信できるようにする許可が含まれています。

Amazon Cognito は、設定を設定するユーザーセッションの AWS 認証情報を使用して、サービスにリンクされたロールを作成します。このセッションの IAM 権限には、iam:CreateServiceLinkedRole アクションが含まれている必要があります。IAM のアクセス許可の詳細については、IAM ユーザーガイドの[AWS 「リソースのアクセス管理」](#)を参照してください。

Amazon Cognito が作成するサービスリンクロールの詳細については、「[Amazon Cognito のサービスリンクロールの使用](#)」を参照してください。

ステップ 4: ユーザープールを設定する

以下のいずれかを使用してユーザープールを設定する場合は、次のステップを完了します。

- E メール送信者として表示されるカスタムの FROM アドレス
- ユーザーが FROM アドレスに送信するメッセージを受信するカスタムの REPLY-TO アドレス
- Amazon SES の設定

Note

検証済み ID が E メールアドレスの場合、Amazon Cognito はデフォルトで、その E メールアドレスを FROM と REPLY-TO の E メールアドレスとして設定します。ただし、検証済み ID がドメインの場合は、FROM の E メールアドレスに値を指定する必要があります。

デフォルトの Amazon Cognito E メール設定とアドレスを使用する場合は、この手順を省略します。

カスタム E メールアドレスを使用するようにユーザープールを設定する方法

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [認証方法] メニューを選択し、[E メール設定] を見つけ、[編集] を選択します。
5. [Edit email configuration] (Eメールの設定) の編集ページで、[Send email from Amazon SES] (Amazon SES からメールを送信する) または[Send email with Amazon Cognito] (Amazon Cognito で E メールを送信する) を選択します。[Send email from Amazon SES] (Amazon SES

からメールを送信する) を選択した場合のみ、[SES Region] (SES リージョン)、[Configuration Set] (設定セット)、および [FROM sender name] (FROM 送信者名) をカスタマイズできます。

6. カスタムの FROM アドレスを使用するには、次の手順を実行します。

- a. [SES Region] (SES リージョン) で、検証済み E メールアドレスが含まれているリージョンを選択します。
- b. [FROM email address] (FROM E メールアドレス) で、E メールアドレスを選択します。Amazon SES で検証済みの E メールアドレスを使用します。
- c. (オプション) [Configuration set] (設定セット) で Amazon SES で使用する設定セットを選択します。この変更を行って保存すると、サービスにリンクされたロールが作成されます。
- d. (オプション) [FROM sender address] (FROM 送信者アドレス) で E メールアドレスを入力します。E メールアドレスだけ、または E メールアドレスと分かりやすい名前を Jane Doe <janedoe@example.com> 形式で指定できます。
- e. (オプション) [REPLY-TO email address] (REPLY-TO E メールアドレス) で、ユーザーが FROM アドレスに送信するメッセージを受信する E メールアドレスを入力します。

7. [Save changes] (変更の保存) をクリックします。

関連トピック

- [E メール検証メッセージのカスタマイズ](#)
- [ユーザー招待メッセージのカスタマイズ](#)

Amazon Cognito ユーザープール用の SMS メッセージ設定

ユーザープールの Amazon Cognito イベントには、Amazon Cognito がユーザーに SMS テキストメッセージを送信する結果となるものがあります。例えば、電話番号の検証を必須とするようにユーザープールを設定している場合、ユーザーがアプリケーションの新しいアカウントにサインアップする、またはパスワードをリセットする場合に Amazon Cognito が SMS テキストメッセージを送信します。SMS テキストメッセージを開始するアクションに応じて、メッセージには検証コード、一時的なパスワード、またはウェルカムメッセージが含まれます。

Amazon Cognito は、SMS テキストメッセージの配信に Amazon Simple Notification Service (Amazon SNS) を使用します。Amazon SNS は、SMS メッセージを渡します AWS End User Messaging SMS。Amazon Cognito 経由で初めてテキストメッセージを送信する場合、は[サンドボックス環境](#) AWS End User Messaging SMS に配置します。サンドボックス環境では、SMS テキ

ストメッセージのアプリケーションをテストできます。サンドボックスでは、メッセージの送信をシミュレートすることだけができます。

 Note

2024 年 11 月、は Amazon SNS SMS メッセージングを AWS に置き換えました AWS End User Messaging SMS。現在、Amazon Cognito コンソールは Amazon SNS リソースを参照しています。ユーザープールは Amazon SNS Publish オペレーションを使用して SMS メッセージを開始します。これはパススルーです AWS End User Messaging SMS。 https://docs.aws.amazon.com/sns/latest/api/API_Publish.htmlしたがって、sms-voice:SendTextMessage ではなく、sns:Publish のアクセス許可を設定する必要があります。

AWS End User Messaging SMS は SMS テキストメッセージの料金を請求します。詳細については、[AWS End User Messaging SMS 料金表](#)を参照してください。

Amazon Cognito は、ユーザーが入力できるコードを含む SMS メッセージをユーザーに送信します。次の表は、SMS メッセージを生成できるイベントを示しています。

メッセージオプション

アクティビティ	API オペレーション:	配信オプション	形式オプション	カスタマイズ可能	メッセージテンプレート
Forgot password	ForgotPassword , AdminResentUserPassword	Email, SMS	code	Yes	検証メッセージ
Invitation	AdminCreateUser	Email, SMS	code	Yes	招待メッセージ
Self-registration	SignUp , ResendConfirmationCode	Email, SMS	code, link	Yes	検証メッセージ
Email address	UpdateUserAttributes ,	Email, SMS	code	Yes	検証メッセージ

アクティビ ティ	API オペレー ション:	配信オブ ション	形式オブ ション	カスタマイ ズ可能	メッセージ テンプレー ト
or phone number verification	AdminUpda teUserAttributes , GetUserAt tributeVerificationCode				
Multi-factor authentica tion (MFA)	AdminInit iateAuth , InitiateAuth	Email ¹ , SMS, authenticator app	code	Yes ²	MFA メッ セージ
One-time password authentica tion (OTP)	AdminInit iateAuth , InitiateAuth	Email ¹ , SMS	code	Yes	MFA メッ セージ ³

¹ エッセンシャル以上の[機能プラン](#)と [Amazon SES E メール設定](#)が必要です。

² SMS メッセージおよび E メールメッセージの場合。

³ ユーザープールで MFA が必須またはオプションである場合にのみ、MFA メッセージテンプレートをカスタマイズできます。MFA が非アクティブの場合、Amazon Cognito はデフォルトのテンプレートを使用してワンタイムパスワードを送信します。

AWS End User Messaging SMS は SMS メッセージに対して課金します。詳細については、[AWS End User Messaging SMS 料金表](#)を参照してください。

MFA の詳細については、「[SMS メッセージ MFA と E メールメッセージ MFA](#)」を参照してください。

Amazon Cognito は、1 つの送信先に対する短期間での E メールや SMS メッセージの追加配信を防止する場合があります。ユーザープールが影響を受けていると思われる場合は、[メッセージ配信エラーのログ](#)を設定および確認してから、アカウントチームにお問い合わせください。

ベストプラクティス

世界中で一方的な SMS トラフィックの量が多いため、一部の政府は SMS メッセージの送信者と受信者の間に障壁を課しています。SMS メッセージを MFA やユーザーアップデートに使用する場合は、メッセージが確実に送られるように追加の手順を踏む必要があります。また、ユーザーが住んでいる可能性のある国の SMS メッセージ関連の規制を監視し、SMS メッセージの設定を最新の状態に保つ必要があります。詳細については、「[AWS End User Messaging SMS ユーザーガイド](#)」の「[SMS および MMS の国別の機能と制限](#)」を参照してください。

SMS メッセージを使用してユーザーを認証および検証することは、セキュリティ上のベストプラクティスではありません。電話番号は所有者が変わる場合があり、ユーザーの MFA の所有している要素を確実に表さないかもしれません。代わりに、アプリまたはサードパーティーの IdP で TOTP MFA を実装してください。また、[カスタム認証チャレンジの Lambda トリガー](#) を使って、追加のカスタム認証要素を作成することもできます。

SMS メッセージ配信アーキテクチャの保護については、以下のリンクを参照してください。

- [Amazon Cognito ユーザープールを使用したユーザーサインアップ詐欺や SMS ポンピングのリスクを軽減する](#)
- [SMS Pumping に対する防御: AWS 人為的に拡張されたトラフィックとの戦いに役立つ新機能](#)

Amazon Cognito ユーザープールでの SMS メッセージングの初回セットアップ

Amazon Cognito は Amazon SNS を間接的に使用して AWS End User Messaging SMS、ユーザープールから SMS メッセージを送信します。[カスタム SMS 送信者の Lambda トリガー](#) を使用して、独自のリソースを使用し、SMS メッセージを送信することもできます。特定の で SMS テキストメッセージを初めてセットアップすると AWS リージョン、はそのリージョン AWS アカウントの SMS サンドボックスに AWS End User Messaging SMS を配置します。AWS End User Messaging SMS はサンドボックスを使用して不正使用を防止し、コンプライアンス要件を満たします。AWS アカウントがサンドボックスにある場合、はいくつかの[制限](#) AWS End User Messaging SMS を適用します。例えば、発信元 ID がある場合、最大 10 個の検証済み送信先番号にテキストメッセージを送信したり、発信元 ID なしでメッセージの送信をシミュレートしたりできます。がサンドボックスに AWS アカウント残っている間は、本番環境で SMS メッセージを送信しないでください。サンドボックスに置かれている間、Amazon Cognito はユーザーの電話番号にメッセージを送信できません。

トピック

- [Amazon Cognito が SMS メッセージを送信するために使用できる IAM ロールを準備する AWS End User Messaging SMS](#)
- [SMS メッセージの AWS リージョン を選択する](#)
- [米国の電話番号に SMS メッセージを送信するための発信元 ID を取得する](#)
- [SMS サンドボックスに置かれていることを確認する](#)
- [アカウントをサンドボックスから移動する](#)
- [でシミュレーター番号または検証済み電話番号を使用する AWS End User Messaging SMS](#)
- [Amazon Cognito でユーザープールのセットアップを完了する](#)

Amazon Cognito が SMS メッセージを送信するために使用できる IAM ロールを準備する AWS End User Messaging SMS

ユーザープールから SMS メッセージを送信すると、Amazon Cognito はアカウントの IAM ロールを引き受けます。Amazon Cognito は、そのロールに割り当てられた `sns:Publish` アクセス許可を使用して、ユーザーに SMS メッセージを送信します。Amazon Cognito コンソールでは、ユーザープールの [認証方法] メニューの [SMS] の下で [IAM ロールの選択] を設定するか、この選択をユーザープール作成ウィザードで行うことができます。

次の IAM ロール信頼ポリシーの例では、Amazon Cognito ユーザープールに、ロールを引き受ける制限付きの機能を付与しています。Amazon Cognito は、次の条件を満たす場合にのみロールを引き受けることができます。

- `assume-role` オペレーションは、`aws:SourceArn` 条件のユーザープールに代わって行われます。
- `assume-role` オペレーションは、`aws:SourceAccount` 条件によって設定された AWS アカウントのユーザープールに代わって行われます。
- `assume-role` オペレーションには、`sts:externalId` 条件の外部 ID が含まれます。

JSON

```
{
  "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-EXAMPLE2222",
            "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:cognito-idp:us-west-2:111122223333:userpool/us-west-2_EXAMPLE"
        }
    }
}
]
}

```

条件 `aws:SourceArn` の値には、正確な [ユーザープール ARN](#) またはワイルドカード ARN を指定できます。AWS マネジメントコンソール または `DescribeUserPool` API リクエストを使用して、ユーザープールの ARNs を検索します。 [DescribeUserPool](#)

[多要素認証](#)用の SMS メッセージを送信するには、IAM ロールの信頼ポリシーに `sts:ExternalId` 条件が含まれている必要があります。この条件の値は、ユーザープールの [SmsConfiguration](#) の `ExternalId` プロパティと一致する必要があります。Amazon Cognito コンソールでユーザープールの作成プロセス中に IAM ロールを作成すると、Amazon Cognito は、ロールとユーザープール設定で外部 ID をセットアップします。これは、既存の IAM ロールを使用する場合には適用されません。

[UpdateUserPool](#) API リクエストのユーザープール `ExternalId` パラメータを更新したうえで、IAM ロールの信頼ポリシーを同じ値の `sts:externalId` 条件で更新する必要があります。API を使用して元の設定を保持するようにユーザープールを更新する方法については、「[ユーザープールとアプリケーションクライアントの設定更新](#)」を参照してください。

IAM ロールおよび信頼ポリシーの詳細については、「AWS Identity and Access Management ユーザーガイド」の「[ロールに関する用語と概念](#)」を参照してください。

SMS メッセージの AWS リージョン を選択する

Note

の SMS メッセージが で管理 AWS されるようになりました[AWS End User Messaging SMS](#)。

一部の では AWS リージョン、Amazon Cognito SMS メッセージに使用する Amazon SNS リソースを含むリージョンを選択できます。Amazon Cognito アジアパシフィック (ソウル) を除く Amazon Cognito が利用可能なすべての AWS リージョンで、ユーザープールを作成した AWS リージョンで Amazon SNS リソースを使用できます。リージョンを選択したときに SMS メッセージングを高速かつ信頼性の高いものにするには、ユーザープールと同じリージョンの Amazon SNS リソースを使用します。

新規ユーザープールウィザードのメッセージ配信の設定ステップで、SMS リソースのリージョンを選択します。既存のユーザープールの [認証方法] メニューで [SMS] の下にある [編集] を選択することもできます。

起動時に、一部の AWS リージョン Amazon Cognito は代替リージョンに Amazon SNS リソースを含む SMS メッセージを送信しました。希望するリージョンを設定するには、ユーザープールの [SmsConfigurationType](#) オブジェクトの `SnsRegion` パラメータを使用します。次の表から Amazon Cognito リージョンに Amazon Cognito ユーザープールリソースをプログラムによって作成し、`SnsRegion` パラメータを提供しない場合、ユーザープールはレガシー Amazon SNS 代替リージョンで Amazon SNS リソースを使用して SMS メッセージを送信します。

アジアパシフィック (ソウル) の Amazon Cognito ユーザープール AWS リージョン は、アジアパシフィック (東京) リージョンの Amazon SNS 設定を使用する必要があります。

Amazon SNS (経由 AWS End User Messaging SMS) は、すべての新しいアカウントの使用量クォータを 1 か月あたり 1.00 USD (USD) に設定します。Amazon Cognito で使用する で支出制限を引き上げ AWS リージョン た可能性があります。AWS リージョン Amazon SNS SMS メッセージの を変更する前に、AWS サポートセンターでクォータ引き上げケースを開いて、新しいリージョンの制限を増やします。詳細については、「[AWS End User Messaging SMS ユーザーガイド](#)」の [AWS End User Messaging SMS 「MMS および音声サンドボックスから本番環境への移行」](#) を参照してください。

次の表の任意の Amazon Cognito リージョンの SMS メッセージを、対応する SMS メッセージリージョンの AWS End User Messaging SMS リソースで送信できます。

Amazon Cognito リージョン	SMS メッセージリージョン
米国東部 (オハイオ)	米国東部 (オハイオ)、米国東部 (バージニア北部)
米国東部 (バージニア北部)	米国東部 (バージニア北部)
米国西部 (北カリフォルニア)	米国西部 (北カリフォルニア)
米国西部 (オレゴン)	米国西部 (オレゴン)
カナダ (中部)	カナダ (中部)、米国東部 (バージニア北部)
カナダ西部 (カルガリー)	カナダ西部 (カルガリー)
メキシコ (中部)	メキシコ (中部)
欧州 (フランクフルト)	欧州 (フランクフルト)、欧州 (アイルランド)
欧州 (ロンドン)	欧州 (ロンドン)、欧州 (アイルランド)
欧州 (アイルランド)	欧州 (アイルランド)
欧州 (パリ)	欧州 (パリ)
欧州 (ストックホルム)	欧州 (ストックホルム)
欧州 (ミラノ)	欧州 (ミラノ)
欧州 (スペイン)	欧州 (スペイン)
欧州 (チューリッヒ)	欧州 (チューリッヒ)
アジアパシフィック (マレーシア)	アジアパシフィック (シンガポール)
アジアパシフィック (タイ)	アジアパシフィック (ムンバイ)
アジアパシフィック (ムンバイ)	アジアパシフィック (ムンバイ)、アジアパシフィック (シンガポール)
アジアパシフィック (ハイデラバード)	アジアパシフィック (ハイデラバード)

Amazon Cognito リージョン	SMS メッセージリージョン
アジアパシフィック (香港)	アジアパシフィック (シンガポール)
アジアパシフィック (ソウル)	アジアパシフィック (東京)
アジアパシフィック (シンガポール)	アジアパシフィック (シンガポール)
アジアパシフィック (シドニー)	アジアパシフィック (シドニー)
アジアパシフィック (東京)	アジアパシフィック (東京)
アジアパシフィック (ジャカルタ)	アジアパシフィック (ジャカルタ)
アジアパシフィック (大阪)	アジアパシフィック (大阪)
アジアパシフィック (メルボルン)	アジアパシフィック (メルボルン)
中東 (バーレーン)	中東 (バーレーン)
中東 (アラブ首長国連邦)	中東 (アラブ首長国連邦)
南米 (サンパウロ)	南米 (サンパウロ)
イスラエル (テルアビブ)	イスラエル (テルアビブ)
アフリカ (ケープタウン)	アフリカ (ケープタウン)

米国の電話番号に SMS メッセージを送信するための発信元 ID を取得する

米国の電話番号に SMS テキストメッセージを送信する場合は、SMS サンドボックステスト環境または本番環境を構築するかどうかにかかわらず、発信元 ID を取得する必要があります。

米国の通信事業者は、米国の電話番号にメッセージを送信する場合に発信元 ID を義務付けています。発信元 ID をまだ取得していない場合は、取得する必要があります。発信元 ID を取得する方法については、「AWS End User Messaging SMS ユーザーガイド」の「[電話番号をリクエストする](#)」を参照してください。

同じ に複数の発信元 ID がある場合 AWS リージョン、 は、ショートコード、10DLC、通話料無料番号の優先順位で発信元 ID タイプ AWS End User Messaging SMS を選択します。この優先順位を

変更することはできません。詳細については、「[AWS End User Messaging SMS に関するよくある質問](#)」を参照してください。

SMS サンドボックスに置かれていることを確認する

次の手順を使用して SMS サンドボックスに移動していることを確認します。本番稼働 AWS リージョン 用 Amazon Cognito ユーザープールがある ごとに繰り返します。

Amazon Cognito コンソールで SMS サンドボックスステータスを確認します。

SMS サンドボックスに置かれていることを確認する

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから存在するユーザープールを 1 つ選択します。
4. [認証方法] メニューを選択します。
5. [SMS configuration] (SMS 設定) セクションで、[Move to Amazon SNS production environment] (Amazon SNS 本番環境に移行する) を展開します。アカウントが SMS サンドボックスに置かれている場合は、以下のメッセージが表示されます。

SMS メッセージの設定を完了するための AWS のサービス 依存関係を設定する

このメッセージが表示されない場合は、アカウント内での SMS メッセージのセットアップが既に実行されています。「[Amazon Cognito でユーザープールのセットアップを完了する](#)」へ進んでください。

6. [Amazon SNS 本番環境に移行] で [Amazon SNS](#) リンクを選択します。これにより、新しいタブで Amazon SNS コンソールが開きます。
7. サンドボックス環境に置かれていることを確認します。コンソールメッセージには、サンドボックスのステータスと AWS リージョン、次のように表示されます。

This account is in the SMS sandbox in US East (N. Virginia).

アカウントをサンドボックスから移動する

本番稼働でアプリを使用するには、アカウントを SMS サンドボックスから本番環境に移動します。Amazon Cognito で使用する AWS End User Messaging SMS リソースを含む発信元 ID をに設定 AWS リージョン したら、 が SMS サンドボックスに AWS アカウント 残っている間、米国の電

話番号を確認できます。本番環境にいる場合は、SMS メッセージを送信する前にユーザーの電話番号を検証する必要はありません。

コンソールまたは Amazon SNS AWS End User Messaging SMS コンソールからサンドボックスを終了するリクエストを作成できます。詳細な手順については、「AWS End User Messaging SMS ユーザーガイド」の「[SMS サンドボックスからの移動](#)」を参照してください。

でシミュレーター番号または検証済み電話番号を使用する AWS End User Messaging SMS SMS サンドボックスからアカウントを移動した場合は、このステップをスキップします。

サンドボックス内に残っていて、発信元番号を設定している場合は、検証済みの送信先番号にメッセージを送信できます。検証済みの送信先を設定するには、「AWS End User Messaging SMS ユーザーガイド」の「[検証済みの送信先電話番号を追加する](#)」を参照してください。

シミュレートした送信元と送信先を使用してメッセージを送信することもできます。シミュレーターメッセージはログを生成しますが、通信事業者ネットワーク経由で送信されることはありません。[\[ショートカット\]](#) メニューから、[SMS シミュレーターで SMS 送信をテストする] を選択します。詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[シミュレーター電話番号](#)」を参照してください。

Amazon Cognito でユーザープールのセットアップを完了する

ユーザープールを作成または[編集](#)していたブラウザタブに戻ります。手順を完了します。ユーザープールに SMS 設定を正常に追加すると、Amazon Cognito は内部の電話番号にテストメッセージを送信して、設定が機能することを確認します。Amazon SNS は、テスト SMS メッセージごとに料金を請求します。

Amazon Cognito ユーザープールのセキュリティ機能を使用する

ネットワーク侵入、パスワードの推測、ユーザーのなりすまし、悪意のあるサインアップやサインインからアプリケーションを保護する場合があります。Amazon Cognito ユーザープールのセキュリティ機能の設定は、セキュリティアーキテクチャの重要なコンポーネントである可能性があります。アプリケーションのセキュリティは、「責任共有モデル」で説明されているように、お客様の責任「クラウド内のセキュリティ」です。AWS <https://aws.amazon.com/compliance/shared-responsibility-model/> この章のツールは、これらの目標に沿ったアプリケーションセキュリティ設計の機能に役立ちます。

ユーザープールを設定するときに行う必要がある重要な決定は、パブリックサインアップとパブリックサインインを許可するかどうかです。機密クライアント、管理上の作成とユーザーの確認、ドメイン

ンのないユーザープールなどのユーザープールオプションの中には、インターネット経由の攻撃の程度が小さいものがあります。ただし、一般的なユースケースは、インターネット上のすべてのユーザーからサインアップを受け入れ、すべてのオペレーションをユーザープールに直接送信するパブリッククライアントです。どの設定でも、特にこうしたパブリック設定の場合は、セキュリティ機能を念頭に置いてユーザープールの計画とデプロイを行うことをお勧めします。セキュリティが不十分な場合も、不要なソースが新しいアクティブユーザーを作成したり、既存のユーザーを悪用しようとしたりすると、AWS 請求に影響する可能性があります。

MFA と脅威保護は、[ローカルユーザー](#)に適用されます。サードパーティー IdP は、[フェデレーションユーザー](#)のセキュリティ体制を担当します。

ユーザープールのセキュリティ機能

多要素認証 (MFA)

ユーザープールのサインインを確認するために、ユーザープールが E メール (エッセンシャル機能プランまたはプラス機能プランの場合)、SMS メッセージ、または Authenticator アプリケーションを使用して送信するコードをリクエストします。

脅威保護

リスクの指標に関してサインインをモニタリングして、MFA を適用するか、またはサインインをブロックします。アクセストークンにカスタムクレームとスコープを追加します。MFA コードを E メールで送信します。

AWS WAF ウェブ ACLs

[ユーザープールエンドポイントと認証 API](#) への受信トラフィックについて、ネットワークレイヤーとアプリケーションレイヤーでの望ましくないアクティビティがないか検査します。

大文字と小文字の区別

E メールアドレスまたは任意のユーザー名が、文字の大文字と小文字の区別を除き、他のユーザーと同じになるユーザーの作成を防止します。

削除保護

自動システムがユーザープールを誤って削除しないようにします。AWS マネジメントコンソールでのユーザープールの削除を追加で確認する必要があります。

ユーザー存在エラー

ユーザープール内の既存のユーザー名とエイリアスの開示を防止します。ユーザー名が有効かどうかにかかわらず、認証に失敗した場合に一般的なエラーを返します。

トピック

- [ユーザープールに MFA を追加します](#)
- [脅威保護を備えた高度なセキュリティ](#)
- [AWS WAF ウェブ ACL をユーザープールに関連付ける](#)
- [ユーザープールの大文字と小文字の区別](#)
- [ユーザープールの削除保護](#)
- [ユーザー存在エラー応答の管理](#)

ユーザープールに MFA を追加します

これは、一般的にはユーザー名とパスワードになっている、初期設定の知っていることの認証要素に、持っていることの認証要素を追加するものです。パスワードを主要な認証要素とするユーザーをサインインさせるには、SMS テキストメッセージ、E メールメッセージ、または時間ベースのワンタイムパスワード (TOTP) を追加要素として選択できます。

多要素認証 (MFA) は、アプリケーション内の[ローカルユーザー](#)のセキュリティを向上させます。[フェデレーションユーザー](#)の場合、Amazon Cognito はすべての認証プロセスを IdP に委任するので、追加の認証要素を提供しません。

Note

新しいユーザーがアプリに初めてサインインするときに、Amazon Cognito は OAuth 2.0 トークンを発行します。これは、ユーザープールに MFA が必要な場合でも同様です。ユーザーが初めてサインインするときの 2 番目の認証要因は、Amazon Cognito が送信する検証メッセージの確認です。ユーザープールに MFA が必要な場合、Amazon Cognito は、最初のログイン試行後の各サインイン試行で使用する追加のサインイン係数を登録するようユーザーに要求します。

アダプティブ認証では、リスクレベルの上昇に対応して、追加の要素認証を要求するようにユーザープールを設定できます。アダプティブ認証をユーザープールに追加するには、「[脅威保護を備えた高度なセキュリティ](#)」を参照してください。

ユーザープールの MFA が required に設定されている場合は、すべてのユーザーはサインインするために MFA を完了する必要があります。サインインするために、各ユーザーには、少なくとも 1 つの MFA 要素が必要です。MFA を必須にする場合は、ユーザーオンボーディングに MFA セットアップを含め、ユーザープールでユーザーのサインインを許可する必要があります。

MFA を必須に設定すると、マネージドログインは MFA を設定することをユーザーに求めます。ユーザープールで MFA をオプションとして設定すると、マネージドログインはユーザーに設定を求めません。オプションの MFA を使用するには、ユーザーに MFA を設定するかどうかを選択するようプロンプトを表示し、API 入力を案内して追加のサインイン要素を確認するインターフェースをアプリに構築する必要があります。

トピック

- [ユーザープールの MFA について知っておくべきこと](#)
- [ユーザー MFA 設定](#)
- [ユーザー実行時の MFA ロジックの詳細](#)
- [ユーザープールでの多要素認証を設定する](#)
- [SMS メッセージ MFA と E メールメッセージ MFA](#)
- [TOTP ソフトウェアトークン MFA](#)

ユーザープールの MFA について知っておくべきこと

MFA を設定する前に、次の点を考慮します。

- ユーザーは MFA を使用するか、またはパスワードなしの要素を使用してサインインできます。
 - [ワンタイムパスワード](#)または[パスキー](#)をサポートするユーザープールでは、MFA を必須に設定することはできません。
 - ユーザープールで MFA が必須である場合、WEB_AUTHN、EMAIL_OTP、または SMS_OTP を AllowedFirstAuthFactors に追加することはできません。Amazon Cognito コンソールでは、選択ベースのサインインのオプションを編集してパスワードなしの要素を含めることはできません。
 - [選択ベースのサインイン](#)では、ユーザープールで MFA が必須である場合にのみ、すべてのアプリケーションクライアントに PASSWORD 要素と PASSWORD_SRP 要素を提供します。ユーザー名パスワードフローの詳細については、このガイドの「認証」章で「[永続的なパスワードによるサインイン](#)」と「[永続的なパスワードと安全なペイロードによるサインイン](#)」を参照してください。
 - MFA がオプションであるユーザープールでは、MFA 要素を設定したユーザーは、選択ベースのサインインでユーザー名パスワード認証フローを使用してのみサインインできます。これらのユーザーは、すべての[クライアントベースのサインイン](#)フローを利用できます。

次の表は、ユーザープールの MFA 設定とユーザーによる MFA 要素の設定が、パスワードなしの要素を使用したユーザーのサインインに与える影響を示しています。

ユーザープールの MFA 設定	ユーザーの MFA ステータス	Webauthn/OTP が利用可能	パスワードによるサインイン後に MFA の設定を求められる	WebAuthn/OTP でサインイン可能
必須	Configured	いいえ	あり	なし
必須	未設定	いいえ	いいえ (サインインできない)	いいえ
オプションです。	Configured	WebAuthn は設定できるが、パスキーではサインインできない	はい	なし
オプションです。	未設定	はい	なし	はい
オフ	いずれか	はい	なし	はい

- ユーザーが希望する MFA の方法は、パスワードの復旧に使用できる方法に影響します。希望する MFA を E メールメッセージにしたユーザーは、パスワードリセットコードを E メールで受信できません。希望する MFA を SMS メッセージにしたユーザーは、パスワードリセットコードを SMS で受信できません。

希望するパスワードリセット方法の対象にユーザーがいない場合、[\[パスワード復旧\]](#) 設定で代替オプションを提供する必要があります。例えば、復旧メカニズムで E メールが最優先事項になっており、E メール MFA がユーザープールのオプションである場合があります。この場合、SMS メッセージによるアカウント復旧を 2 番目のオプションとして追加するか、管理 API オペレーションを使用してこれらのユーザーのパスワードをリセットします。

Amazon Cognito は、有効な復旧方法を持たないユーザーからのパスワードリセットリクエストに対して、InvalidParameterException エラーレスポンスで応答します。

[UpdateUserPool](#) のリクエスト本文の例は、AccountRecoverySetting を示しています。E メールメッセージによるパスワードのリセットが利用できない場合、ユーザーはここにフォールバックして、SMS メッセージによる復旧を行うことができます。

- ユーザーは、MFA とパスワードのリセットコードを、同じ E メールアドレスや電話番号で受け取ることはできません。E メールメッセージのワンタイムパスワード (OTP) を MFA に使用する場合、アカウントの復旧には SMS メッセージを使用する必要があります。SMS メッセージの OTP を MFA に使用する場合、アカウントの復旧には E メールメッセージを使用する必要があります。MFA を使用するユーザープールでは、属性として E メールアドレスがあっても電話番号がないか、電話番号があっても E メールアドレスがない場合、ユーザーはセルフサービスのパスワード復旧を完了できない可能性があります。

この設定でユーザーがユーザープールのパスワードをリセットできない状況を防ぐには、email および phone_number [属性を必須](#)に設定します。別の方法として、ユーザーのサインアップ時や管理者によるユーザープロフィールの作成時に、これらの属性を常に収集して設定するようにプロセスを設定することもできます。ユーザーが両方の属性を持っている場合、Amazon Cognito は、ユーザーの MFA 要素ではない送信先にパスワードリセットコードを自動的に送信します。

- ユーザープールで MFA を有効にし、2 番目の要素として SMS メッセージまたは E メールメッセージを選択すると、Amazon Cognito で検証していない電話番号または E メール属性にメッセージを送信できます。ユーザーが MFA を完了すると、Amazon Cognito は、phone_number_verified 属性または email_verified 属性を true に設定します。
- MFA コードの提示に 5 回失敗すると、Amazon Cognito は [サインイン試行の失敗時におけるロックアウト動作](#) で説明した指数関数的タイムアウトロックアウトプロセスを開始します。
- アカウントがユーザープールの Amazon Simple Notification Service (Amazon SNS) リソース AWS リージョン を含む の SMS サンドボックスにある場合は、SMS メッセージを送信する前に Amazon SNS で電話番号を確認する必要があります。詳細については、「[Amazon Cognito ユーザープール用の SMS メッセージ設定](#)」を参照してください。
- 脅威保護で検出されたイベントに応じてユーザーの MFA ステータスを変更するには、Amazon Cognito ユーザープールコンソールで MFA を有効にしてオプションとして設定します。詳細については、「[脅威保護を備えた高度なセキュリティ](#)」を参照してください。
- E メールメッセージと SMS メッセージでは、各ユーザーに E メールアドレスと電話番号の属性が必要です。ユーザープールでは、email または phone_number を必須の属性として設定できます。この場合、ユーザーは電話番号を指定しない限りサインアップを完了できません。これらの属性について必要に応じた設定を行わずに、E メールメッセージ MFA または SMS メッセージ MFA を実行する場合は、サインアップ時に E メールアドレスまたは電話番号の入力を求めます。ベス

トプラクティスとして、ユーザープールを設定して、[これらの属性を検証する](#)ようにユーザーに自動的にメッセージを送信します。

Amazon Cognito は、ユーザーが SMS メッセージまたは E メールメッセージで仮コードを受信し、[VerifyUserAttribute](#) API リクエストでそのコードを返した場合、その電話番号または E メールアドレスを検証済みとしてカウントします。代わりに、チームは、電話番号を設定し、[AdminUpdateUserAttributes](#) API リクエストを実行する管理用アプリケーションを使用して検証済みとしてマークできます。

- MFA を必須に設定し、複数の認証要素をアクティブ化した場合、Amazon Cognito は新しいユーザーに、使用する MFA 要素を選択するよう求めます。ユーザーは、SMS メッセージ MFA をセットアップするための電話番号と、E メールメッセージ MFA をセットアップするための E メールアドレスを持っている必要があります。使用可能なメッセージベースの MFA に属性が定義されていない場合、Amazon Cognito は TOTP MFA を設定するようにユーザーに求めます。MFA 要素 (SELECT_MFA_TYPE) を選択し、選択した要素 (MFA_SETUP) を設定するように求めるプロンプトが、[InitiateAuth](#) および [AdminInitiateAuth](#) の API オペレーションへのチャレンジレスポンスとして表示されます。

ユーザー MFA 設定

ユーザーは複数の MFA 要素を設定できます。アクティブにできるのは 1 つだけです。ユーザープール設定またはユーザープロンプトで、ユーザーの有効な MFA 設定を選択できます。ユーザープールの設定と独自のユーザーレベルの設定が以下の条件を満たす場合、ユーザープールは MFA コードをユーザーに要求します。

1. ユーザープールで MFA をオプションまたは必須に設定している。
2. ユーザーは、有効な email 属性または phone_number 属性を持っているか、Authenticator アプリケーションを TOTP 用にセットアップ済みである。
3. 少なくとも 1 つの MFA 要素がアクティブである。
4. 1 つの MFA 要素を優先として設定している。

サインインと MFA に同じ要素を使用しない

ユーザープールは、1 つのサインイン要素を一部のユーザーまたはすべてのユーザーが使用できる唯一のサインインおよび MFA オプションにする方法で設定できます。この結果は、プライマリサインインのユースケースが E メールメッセージまたは SMS メッセージワンタイムパスワード (OTPs) で

ある場合に発生する可能性があります。ユーザーが優先する MFA は、次の条件でサインインと同じタイプの要素である場合があります。

- ユーザープールには MFA が必要です。
- E メールと SMS OTP は、ユーザープールでサインインオプションと MFA オプションを使用できます。
- ユーザーは E メールまたは SMS メッセージ OTP でサインインします。
- E メールアドレス属性はありますが、電話番号属性はありません。または、電話番号属性はありますが、E メールアドレス属性はありません。

このシナリオでは、ユーザーは E メール OTP でサインインし、E メール OTP で MFA を完了できます。このオプションは、MFA の必須関数をキャンセルします。ワンタイムパスワードでサインインするユーザーは、MFA とは異なる配信方法をサインインに使用できる必要があります。ユーザーに SMS と Eメールの両方のオプションがある場合、Amazon Cognito は別の要素を自動的に割り当てます。たとえば、ユーザーが E メール OTP でサインインする場合、希望する MFA は SMS OTP です。

ユーザープールがサインインと MFA の両方で OTP 認証をサポートしている場合に、次の手順を実行して同要素認証に対処します。

1. サインイン要因として E メールと SMS OTP の両方を有効にします。
2. E メールと SMS OTP の両方を MFA 要因として有効にします。
3. 収集

ユーザープールの設定と MFA オプションへの影響

ユーザープールの設定は、ユーザーが選択できる MFA の方法に影響します。以下のいくつかのユーザープール設定が、ユーザーによる MFA の設定に影響します。

- Amazon Cognito コンソールの [サインイン] メニューの [多要素認証] 設定で、MFA をオプションまたは必須に設定するか、オフにすることができます。この設定と同等の API は、`CreateUserPool`、`UpdateUserPool`、および `SetUserPoolMfaConfig` の [MfaConfiguration](#) パラメータです。

また、[多要素認証] 設定では、[MFA の方法] 設定によって、ユーザーが設定できる MFA 要素が決まります。この設定と同等の API は [SetUserPoolMfaConfig](#) オペレーションです。

- [サインイン] メニューの [ユーザーアカウントの復旧] で、パスワードを忘れたユーザーにユーザープールからメッセージを送信する方法を設定できます。ユーザーの MFA の方法は、パスワードを忘れた場合のコードをユーザープールから配信する方法と同じにすることはできません。パスワードを忘れた場合の配信方法の API パラメータは、CreateUserPool および UpdateUserPool の [AccountRecoverySetting](#) パラメータです。

例えば、復旧オプションが [E メールのみ] の場合、ユーザーは E メール MFA を設定できません。これは、同じユーザープールで E メール MFA を有効にして、復旧オプションを [E メールのみ] に設定することはできないためです。このオプションを [使用可能な場合は E メール、それ以外の場合は SMS] に設定すると、E メールが優先の復旧オプションになりますが、ユーザーが E メールメッセージ復旧の対象でない場合、ユーザープールは SMS メッセージにフォールバックできます。この場合、ユーザーは E メール MFA を優先として設定し、パスワードをリセットするときのみ SMS メッセージを受信できます。

- 使用できるものとして 1 つの MFA の方法のみを設定した場合、ユーザーの MFA 設定を管理する必要はありません。
- SMS をアクティブに設定すると、ユーザープールで SMS メッセージが、使用できる MFA の方法に自動的にになります。

ユーザープール内の独自の Amazon SES リソースと、エッセンシャル機能プランまたはプラス機能プランを使用した、アクティブな [E メール設定](#) により、E メールメッセージは自動的にユーザープールで使用可能な MFA 方法になります。

- ユーザープールで MFA を必須に設定すると、ユーザーは MFA の方法を有効または無効にできません。設定できるのは、希望する方法のみです。
- ユーザープールで MFA をオプションとして設定すると、マネージドログインは、MFA を設定するようユーザーに求めません。ただし、希望する MFA の方法を持っているユーザーには、MFA コードの入力を求めます。
- [脅威保護](#) を有効にし、アダプティブ認証レスポンスをフル機能モードで設定する場合、MFA がユーザープールでオプションになっている必要があります。アダプティブ認証によるレスポンスオプションの 1 つは、サインイン試行にリスクレベルが含まれていると評価されたユーザーに MFA を要求することです。

コンソールの [サインアップ] メニューの [必須属性] 設定は、ユーザーがアプリケーションにサインアップするために E メールアドレスまたは電話番号を入力する必要があるかどうかを決定します。ユーザーが対応する属性を持っている場合、E メールメッセージと SMS メッセージは MFA 要素として適格になります。CreateUserPool の [スキーマ](#) パラメータは、必要に応じて属性を設定します。

- ユーザープールで MFA を必須として設定した場合、ユーザーがマネージドログインを使用してサインインすると、Amazon Cognito はユーザープールで利用可能な MFA 方法から選択するようユーザーに求めます。マネージドログインは、E メールアドレスまたは電話番号の収集と TOTP の設定を処理します。以下の図は、Amazon Cognito がユーザーに提示するオプションの背後のロジックを示しています。

ユーザーの MFA 設定を構成する

アクセストークン認証を使用するセルフサービスモデル、または管理 API オペレーションを使用する管理者管理モデルで、ユーザーの MFA の希望を設定できます。これらのオペレーションは MFA の方法を有効または無効にし、複数の方法のうちのいずれかを、希望するオプションとして設定します。ユーザーが MFA の希望を設定すると、Amazon Cognito はサインイン時に、希望する MFA の方法によるコードを提供するようにユーザーに求めます。希望を設定していないユーザーは、SELECT_MFA_TYPE チャレンジで、希望する方法を選択するよう求めるプロンプトを受け取ります。

- ユーザーセルフサービスモデルまたはパブリックアプリケーションでは、サインインユーザーのアクセストークンで認可された [SetUserMfaPreference](#) によって MFA の設定が決まります。
- 管理者管理アプリケーションまたは機密アプリケーションでは、管理者 AWS 認証情報で承認された [AdminSetUserPreference](#) が MFA 設定を設定します。

また、Amazon Cognito コンソールの [ユーザー] メニューからユーザーの MFA 設定を構成することもできます。Amazon Cognito ユーザープール API オペレーションのパブリック認証モデルと機密認証モデルの詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

ユーザー実行時の MFA ロジックの詳細

ユーザーがサインインするときに実行する手順を決定するために、ユーザープールはユーザーの MFA 設定、[ユーザー属性](#)、[ユーザープールの MFA 設定](#)、[脅威保護](#)アクション、[セルフサービスのアカウントの復旧](#)に関する設定を評価します。次に、ユーザーをサインインさせ、MFA の方法の選択、設定、または入力をユーザーに求めます。MFA の方法を設定する場合、ユーザーは [E メールアドレスまたは電話番号](#)を指定するか、[TOTP Authenticator を登録](#)する必要があります。事前に MFA のオプションを設定し、[優先オプションを登録](#)することもできます。以下の図は、ユーザープールの設定が、初回サインアップ直後のサインイン試行にどのような影響を及ぼすかを詳しく示しています。

[illegible]

す。✔
は、認証の成功とフローの結果を示します。

す。✖
は、認証の失敗を示します。

-
- 1050

2. ユーザー名パスワード認証が成功したら、MFA を必須、オプション、またはオフのいずれにするかを決めます。オフにした場合は、正しいユーザー名とパスワードを入力すると、認証が成功します。 
- a. MFA をオプションにした場合は、ユーザーが TOTP Authenticator を設定済みであるかどうかを確認します。TOTP を設定済みである場合は、TOTP を使用した MFA を求めます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。 
- b. 脅威保護のアダプティブ認証機能により、MFA を設定することをユーザーに求めているかどうかを確認します。MFA が割り当てられていない場合、ユーザーはサインインされます。 
3. MFA が必須であるか、アダプティブ認証が MFA を割り当て済みである場合は、ユーザーが MFA 要素を有効および優先として設定しているかどうかを確認します。設定している場合は、その要素を使用した MFA の入力を求めます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。 
4. ユーザーが MFA 設定を指定していない場合は、ユーザーが TOTP Authenticator を登録済みであるかどうかを確認します。
 - a. ユーザーが TOTP Authenticator を登録済みである場合は、TOTP MFA がユーザープールで使用可能かどうかを確認します (ユーザーが以前に Authenticator を設定した後で、TOTP MFA が無効になっている可能性があります)。
 - b. E メールメッセージ MFA または SMS メッセージ MFA もユーザープールで利用できるかどうかを確認します。
 - c. E メール MFA も SMS MFA も利用できない場合は、ユーザーに TOTP を使用した MFA を求めます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。 
 - d. E メール MFA または SMS MFA が利用可能な場合は、ユーザーが対応する email 属性または phone_number 属性を持っているかどうかを確認します。持っている場合は、セルフサービスのアカウント復旧のプライマリ方法ではなく、MFA が有効になっている属性であれば、ユーザーが使用できます。
 - e. TOTP と利用可能な SMS MFA 要素または E メール MFA 要素を含む MFAS_CAN_SELECT オプションを使用して、SELECT_MFA_TYPE チャレンジに応答するようユーザーに求めます。

- f. SELECT_MFA_TYPE チャレンジに応じて選択した要素の入力をユーザーに求めます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。✔
5. ユーザーが TOTP Authenticator を登録していないか、登録済みでも TOTP MFA が現在無効になっている場合は、ユーザーが email 属性または phone_number 属性を持っているかどうかを確認します。
6. ユーザーが属性として E メールアドレスまたは電話番号のみを持っている場合、その属性が、ユーザープールでパスワードリセット用のアカウント復旧メッセージを送信するために実装する方法と同じであるかどうかを確認します。同じである場合、MFA を必須としているサインインは完了できず、Amazon Cognito はエラーを返します。このユーザーのサインインを有効にするには、復旧用ではない属性を追加するか、TOTP Authenticator を登録する必要があります。✖
- a. 復旧用ではない E メールアドレスや電話番号を利用できる場合は、対応する E メール要素または SMS MFA 要素が有効になっているかどうかを確認します。
- b. 復旧用ではない E メールアドレス属性があり、E メール MFA が有効になっている場合は、EMAIL_OTP チャレンジに応答するよう求められます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。✔
- c. 復旧用ではない電話番号属性があり、SMS MFA が有効になっている場合は、SMS_MFA チャレンジに応答するよう求められます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。✔
- d. 有効な E メール MFA 要素または SMS MFA 要素の対象となる属性がない場合は、TOTP MFA が有効になっているかどうかを確認します。TOTP MFA が無効になっている場合、ユーザーは MFA を必須としているサインインを完了できず、Amazon Cognito はエラーを返します。このユーザーのサインインを有効にするには、復旧用ではない属性を追加するか、TOTP Authenticator を登録する必要があります。✖

 Note

ユーザーが TOTP Authenticator を持っているが TOTP MFA が無効になっている場合、このステップは [いいえ] と評価済みです。

- e. TOTP MFA が有効になっている場合は、MFAS_CAN_SETUP オプションの SOFTWARE_TOKEN_MFA を使用してユーザーに MFA_SETUP チャレンジを提示します。このチャレンジを完了するには、ユーザーの TOTP Authenticator を個別に登録し、"ChallengeName": "MFA_SETUP", "ChallengeResponses": {"USERNAME": "[username]", "SESSION": "[Session ID from VerifySoftwareToken]}" を使用して応答する必要があります。
 - f. ユーザーが [VerifySoftwareToken](#) リクエストからのセッショントークンで MFA_SETUP チャレンジに応答すると、ユーザーは SOFTWARE_TOKEN_MFA チャレンジに応答するように求められます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。 
7. ユーザーが E メールアドレスと電話番号の両方を持っている場合は、どちらの属性がパスワードリセットのアカウント復旧メッセージのプライマリ方法であるかを確認します。
- a. セルフサービスのアカウントの復旧が無効になっている場合は、どちらの属性も MFA に使用できます。E メール MFA 要素と SMS MFA 要素の一方または両方が有効になっているかどうかを確認します。
 - b. 両方の属性が MFA 要素として有効になっている場合は、MFAS_CAN_SELECT のオプション (SMS_MFA と EMAIL_OTP) を使用して SELECT_MFA_TYPE チャレンジに応答するようユーザーに求めます。
 - c. SELECT_MFA_TYPE チャレンジに応じて選択した要素の入力をユーザーに求めます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。 
 - d. 1 つの属性のみが適格な MFA 要素である場合は、残りの要素のチャレンジに応答するようユーザーに求めます。MFA チャレンジに正常に応答すると、ユーザーはサインインされます。 

この結果が生じるのは、以下の場合です。

- i. ユーザーが email 属性と phone_number 属性を持っていて、SMS MFA と E メール MFA が有効であり、アカウント復旧のプライマリ方法が E メールまたは SMS メッセージである場合。
 - ii. ユーザーが email 属性と phone_number 属性を持っていて、SMS MFA または E メール MFA の一方のみが有効であり、セルフサービスのアカウントの復旧が無効になっている場合。
8. ユーザーが TOTP Authenticator を登録しておらず、email 属性も phone_number 属性も持っていない場合は、MFA_SETUP チャレンジに回答するようユーザーに求めます。MFAS_CAN_SETUP のリストには、ユーザープールのすべての有効な MFA 要因のうち、アカウント復旧のプライマリ方法ではないものが表示されます。ユーザーは E メール MFA または TOTP MFA で ChallengeResponses を使用し、このチャレンジに回答できます。SMS MFA を設定するには、電話番号属性を別個に追加し、認証を再開します。

TOTP MFA の場合は、`"ChallengeName": "MFA_SETUP", "ChallengeResponses": {"USERNAME": "[username]", "SESSION": "[Session ID from VerifySoftwareToken]"}` を使用して応答します。

E メール MFA の場合は、`"ChallengeName": "MFA_SETUP", "ChallengeResponses": {"USERNAME": "[username]", "email": "[user's email address]"}` を使用して応答します。

- a. SELECT_MFA_TYPE チャレンジに応じて選択した要素の入力をユーザーに求めます。MFA チャレンジに正常に回答すると、ユーザーはサインインされます。 

ユーザープールでの多要素認証を設定する

MFA を設定するには、Amazon Cognito コンソールを使用するか、[SetUserPoolMfaConfig](#) API オペレーションと SDK メソッドを使用できます。

Amazon Cognito コンソールで MFA を設定する

1. [Amazon Cognito コンソール](#)にサインインします。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [サインイン] メニューを選択します。[多要素認証] を見つけて、[編集] を選択します。
5. ユーザープールで使用する [MFA enforcement] (MFA 強制実行) 方法を選択します。

Edit multi-factor authentication (MFA) [Info](#)

Amazon Cognito has additional authentication factors with SMS messages, email message, 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.

☐ **Email message**
Users can authenticate with a code sent in an email message. Email messages are charged separately by Amazon SES. [Learn more about pricing](#) [↗](#)

[Cancel](#) [Save changes](#)

- a. MFA が必要。ユーザープール内のすべてのユーザーは、さらに SMS、E メール、または時間ベースのワンタイムパスワード (TOTP) コードを追加の認証要素として使用し、サインインする必要があります。
 - b. オプションの MFA。ユーザーに追加のサインイン要素を登録するオプションを提供した場合でも、MFA を設定していないユーザーに依然としてサインインを許可できます。アダプティブ認証を使用している場合は、このオプションを選択してください。アダプティブ認証の詳細については、「[脅威保護を備えた高度なセキュリティ](#)」を参照してください。
 - c. MFA なし。ユーザーは、サインイン要素を追加登録することはできません。
6. アプリケーションでサポートする [MFA methods] (MFA メソッド) を選択します。2 番目の要素として E メールメッセージ、SMS メッセージまたは TOTP 生成認証アプリケーションを設定できます。
 7. 2 番目の要素として SMS テキストメッセージを使用していて、SMS メッセージ用の Amazon Simple Notification Service で使用する IAM ロールを設定していない場合は、コンソールで作成できます。ユーザープールの [認証方法] メニューで、[SMS] を見つけて [編集] を選択します。Amazon Cognito がユーザーに SMS メッセージを送信することを許可する既存のロールを使用することもできます。詳細については、「[IAM ロール](#)」を参照してください。

E メールメッセージを第 2 の要素として使用し、Amazon Simple Email Service (Amazon SES) で E メールメッセージに使用する送信元 ID を設定していない場合は、コンソールで作成します。[SES で E メールを送信] オプションを選択する必要があります。ユーザープールの [認証方法] メニューで、[E メール] を見つけて [編集] を選択します。リストの利用可能な検証済み ID から [送信元の E メールアドレス] を選択します。検証済みドメイン (example.com など) を選択

した場合は、検証済みドメイン (admin-noreply@example.com など) で [送信者の名前] も設定する必要があります。

8. [Save changes] (変更の保存) をクリックします。

SMS メッセージ MFA と E メールメッセージ MFA

SMS MFA メッセージと Eメールの MFA メッセージは、サインインする前にユーザーがメッセージの宛先にアクセスできることを確認します。パスワードだけでなく、元のユーザーの SMS メッセージまたは Eメールの受信トレイにもアクセスできることを確認します。Amazon Cognito は、ユーザーがユーザー名とパスワードを正常に提供した後にユーザープールが送信した短いコードを提供するようにユーザーに要求します。

ユーザーがプロフィールに E メールアドレスまたは電話番号を追加すると、SMS メッセージ MFA および E メールメッセージ MFA に追加の設定は必要ありません。Amazon Cognito は、未検証の E メールアドレスと電話番号にメッセージを送信できます。ユーザーが最初の MFA を完了すると、Amazon Cognito は、その E メールアドレスまたは電話番号を検証済みとしてマークします。

MFA 認証は、MFA を持つユーザーがアプリケーションにユーザー名とパスワードを入力したときに開始されます。アプリケーションは、[InitiateAuth](#) API リクエストまたは [AdminInitiateAuth](#) API リクエストを呼び出す SDK メソッドでこれらの初期パラメータを送信します。API レスポンスの `ChallengeParameters` には、認証コードが送信された場所を示す `CODE_DELIVERY_DESTINATION` 値が含まれます。アプリケーションで、ユーザーに電話を確認するように求め、かつコードの入力要素を含めたフォームを表示します。コードを入力したら、チャレンジレスポンス API リクエストでコードを送信してサインインプロセスを完了します。

MFA を持つユーザーが[マネージドログイン](#)ページでユーザー名とパスワードを使用してサインインすると、ユーザーは MFA コードを入力するよう自動的に求められます。

ユーザープールは、AWS アカウントの Amazon Simple Notification Service (Amazon SNS) リソースを使用して、MFA およびその他の Amazon Cognito 通知に関して SMS メッセージを送信します。同様に、ユーザープールは、アカウントの Amazon Simple Email Service (Amazon SES) リソースを使用して E メールメッセージを送信します。これらのリンクされたサービスでは、メッセージ配信の AWS 請求書に独自のコストが発生します。また、本番稼働ボリュームでメッセージを送信するための追加要件もあります。詳細については、以下のリンクを参照してください。

- [Amazon Cognito ユーザープール用の SMS メッセージ設定](#)
- [ワールドワイド SMS 料金](#)
- [Amazon Cognito ユーザープールの E メール設定](#)

- [Amazon SES の価格設定](#)

SMS メッセージ MFA と E メールメッセージ MFA に関する考慮事項

- ユーザーに E メール MFA でのサインインを許可するには、ユーザープールに次の設定オプションが必要です。
 1. ユーザープールにプラス機能プランまたはエッセンシャル機能プランがある。詳細については、「[ユーザープールの機能プラン](#)」を参照してください。
 2. ユーザープールは、独自の Amazon SES リソースを使用して E メールメッセージを送信する。詳細については、「[Amazon SES の E メール設定](#)」を参照してください。
- MFA コードは、アプリケーションクライアントで設定した認証フローセッションの持続期間の間有効です。

認証フローセッションの持続期間は、Amazon Cognito コンソールの [アプリケーションクライアント] メニューで、アプリケーションクライアントを編集するときに設定します。CreateUserPoolClient または UpdateUserPoolClient API リクエストで認証フローセッション持続期間を設定することも可能です。詳細については、「[認証セッションの例](#)」を参照してください。

- 検証されていない電話番号または E メールアドレスに Amazon Cognito が送信した SMS または E メールメッセージからユーザーがコードを正常に提供すると、Amazon Cognito は、対応する属性を検証済みとしてマークします。
- MFA に関連付けられた電話番号または E メールアドレスの値に対してユーザーがセルフサービスで変更を加えるには、サインインしたうえでアクセストークンを使用してリクエストを認可する必要があります。現在の電話番号または E メールアドレスにアクセスできない場合は、サインインできません。チームは、[AdminUpdateUserAttributes](#) API リクエストで管理者 AWS 認証情報を使用してこれらの値を変更する必要があります。
- ユーザープールで [SMS を設定](#) した後は、使用できる MFA 要素として SMS メッセージを無効にすることはできません。

TOTP ソフトウェアトークン MFA

ユーザープールで TOTP ソフトウェアトークン MFA を設定すると、ユーザーはユーザー名とパスワードでサインインし、TOTP を使用して認証を完了します。ユーザーがユーザー名とパスワードを設定して検証した後、MFA の TOTP ソフトウェアトークンを有効化できます。アプリケーションでユーザーのサインインに Amazon Cognito のマネージドログインを使用する場合、ユーザーはユーザー名とパスワードを送信し、別のサインインページで TOTP パスワードを送信します。

ユーザープールの TOTP MFA は、Amazon Cognito コンソールでアクティブ化すること
も、Amazon Cognito API オペレーションを使用してもかまいません。ユーザープールレベルで
は、[SetUserPoolMfaConfig](#) を呼び出すことで、MFA を設定して TOTP MFA を有効化できます。

 Note

TOTP ソフトウェアトークン MFA がユーザープールに対して有効になっていない場合は、Amazon Cognito で、ユーザーをトークンに関連付けたり、トークンで検証することができません。この場合、ユーザーは `SoftwareTokenMfaNotFoundException` 説明による例外 `Software Token MFA has not been enabled by the userPool` を受診します。ソフトウェアトークン MFA が後にユーザープールに対して非アクティブ化された場合、以前に TOTP トークンに関連付けて検証したユーザーは、引き続き MFA で使用することができます。

ユーザーの TOTP の設定には複数のステップが伴います。ユーザーはシークレットコードを受け取り、ワンタイムパスワードを入力して、このコードを検証します。次に、ユーザーの TOTP MFA を有効にするか、ユーザーの優先 MFA メソッドとして TOTP を設定することができます。

ユーザープールで TOTP MFA を必須に設定した場合、ユーザーがマネージドログインでアプリケーションにサインアップするときに、Amazon Cognito はユーザープロセスを自動化します。Amazon Cognito は、ユーザーに MFA メソッドの選択を促し、Authenticator アプリケーションを設定するための QR コードを表示し、MFA 登録を確認します。ユーザーが SMS と TOTP MFA のどちらかを選択できるようにしたユーザープールでは、Amazon Cognito もユーザーに選択肢を提示します。

 Important

ユーザープールに関連付けられた AWS WAF ウェブ ACL があり、ウェブ ACL のルールが CAPTCHA を提示すると、マネージドログイン TOTP 登録で回復不可能なエラーが発生する可能性があります。CAPTCHA アクションを含み、マネージドログインの TOTP には影響しないルールを作成するには、「[マネージドログイン TOTP MFA 用の AWS WAF ウェブ ACL の設定](#)」を参照してください。AWS WAF ウェブ ACL と Amazon Cognito の詳細については、「[AWS WAF ウェブ ACL をユーザープールに関連付ける](#)」を参照してください。

AWS SDK と [Amazon Cognito ユーザープール API](#) を使用してカスタムビルド UI に TOTP MFA を実装するには、「[ユーザーの TOTP MFA の設定](#)」を参照してください。

MFA をユーザープールに追加するには、「[ユーザープールに MFA を追加します](#)」を参照してください。

TOTP MFA 考慮事項と制約事項

1. Amazon Cognito は、TOTP コードを生成する Authenticator アプリケーションを介してソフトウェアトークン MFA をサポートします。Amazon Cognito はハードウェアベースの MFA をサポートしていません。
2. TOTP を設定していないユーザーに対してユーザープールが TOTP を必要とする場合、ユーザーはワンタイムアクセストークンを受け取り、アプリはそれを使ってユーザーの TOTP MFA を有効化することができます。後続のサインイン試行は、ユーザーが追加の TOTP サインイン要素を登録するまで失敗します。
 - ユーザーが SignUp API オペレーションまたはマネージドログインを使用してユーザープールにサインアップすると、サインアップを完了したときに 1 回限りのトークンを受け取ります。
 - ユーザーを作成し、ユーザーが初期パスワードを設定すると、Amazon Cognito はマネージドログインからユーザーに 1 回限りのトークンを発行します。ユーザーに永続的なパスワードを設定すると、ユーザーが最初にサインインしたときに Amazon Cognito が 1 回限りのトークンを発行します。
 - Amazon Cognito は、[InitiateAuth](#) または [AdminInitiateAuth](#) API オペレーションでサインインする管理者が作成したユーザーには 1 回限りのトークンを発行しません。ユーザーが初期パスワードの設定のチャレンジに成功した後、またはユーザーに永続的なパスワードを設定すると、Amazon Cognito はすぐに MFA の設定をユーザーに要求します。
3. MFA を必須としているユーザープールのユーザーは、ワンタイムアクセストークンを既に受けていても、TOTP MFA を設定していない場合、MFA を設定するまでマネージドログインでサインインできません。アクセストークンの代わりに、[InitiateAuth](#) または [AdminInitiateAuth](#) に対する MFA_SETUP チャレンジの session レスポンス値を、[AssociateSoftwareToken](#) リクエストで使用できます。
4. ユーザーが TOTP を設定した場合は、TOTP が後でユーザープールに対して無効にされた場合でも、その TOTP を MFA に使用できます。
5. Amazon Cognito は、HMAC-SHA1 ハッシュ関数を使用してコードを生成する認証アプリケーションからの TOTP のみを受け入れます。SHA-256 ハッシュで生成されたコードは Code mismatch エラーを返します。

ユーザーの TOTP MFA の設定

ユーザーが最初にサインインすると、アプリはワンタイムアクセストークンを使用して TOTP プライベートキーを生成し、テキスト形式または QR コード形式でユーザーに提示します。ユーザーは Authenticator アプリケーションを設定し、その後のサインイン試行の TOTP を提供します。アプリケーションまたはマネージドログインは、この TOTP を MFA チャレンジレスポンスで TOTP を Amazon Cognito に提示します。

状況によっては、マネージドログインにより、新規ユーザーに TOTP Authenticator を設定するように求めます。詳細については、「[ユーザー実行時の MFA ロジックの詳細](#)」を参照してください。

トピック

- [TOTP ソフトウェアトークンを関連付ける](#)
- [TOTP トークンを検証](#)
- [TOTP MFA でのサインイン](#)
- [TOTP トークンを削除](#)

TOTP ソフトウェアトークンを関連付ける

TOTP トークンを関連付けるには、ワンタイムパスワードで検証する必要があるシークレットコードをユーザーに送信する必要があります。トークンの関連付けには 3 つのステップが必要です。

1. ユーザーが TOTP ソフトウェアトークン MFA を選択したら、[AssociateSoftwareToken](#) を呼び出して、一意に生成された共有シークレットキーコードをユーザーアカウントに返します。AssociateSoftwareToken は、アクセストークンまたはセッション文字列を使用して承認できます。
2. アプリは、プライベートキーまたはプライベートキーから生成した QR コードをユーザーに提示します。ユーザーは、Google Authenticator などの TOTP 生成アプリにプライベートキーを入力する必要があります。入力するには、アプリケーションでプライベートキーから生成した QR コードをスキャンするか、手動で入力します。
3. ユーザーがキーを入力するか、Google Authenticator などの認証システムアプリケーションに QR コードをスキャンすると、アプリがコードの生成を開始します。

TOTP トークンを検証

次に、TOTP トークンを検証します。以下のように、ユーザーからサンプルコードをリクエストし、Amazon Cognito サービスに提供して、ユーザーが TOTP コードを正常に生成していることを確認します。

1. アプリは、ユーザーが認証システムアプリケーションを適切に設定したことを示すコードの入力をユーザーに促します。
2. ユーザーの認証システムアプリケーションは、一時的なパスワードを表示します。認証システムアプリケーションは、ユーザーに与えたシークレットキーに基づいてパスワードを作成します。
3. ユーザーは一時パスワードを入力します。アプリは、[VerifySoftwareToken](#) API リクエストで一時パスワードを Amazon Cognito に渡します。
4. Amazon Cognito は、ユーザーに関連付けられたシークレットキーを保持し、TOTP を生成し、ユーザーが指定したシークレットキーと比較します。一致した場合は、VerifySoftwareToken は SUCCESS レスポンスを返します。
5. Amazon Cognito は TOTP 要素をユーザーに関連付けます。
6. VerifySoftwareToken オペレーションが ERROR レスポンスを返した場合は、ユーザーのクロックが正しいこと、およびリトライの最大回数を超えていないことを確認します。Amazon Cognito は、試行の前後 30 秒以内の TOTP トークンを受け入れ、マイナークロックキューを考慮します。問題が解決したら、VerifySoftwareToken オペレーションをもう一度試してください。

TOTP MFA でのサインイン

この時点で、ユーザーは時間ベースのワンタイムパスワードを使用したサインインを行います。以下はその手順です。

1. ユーザーはユーザー名とパスワードを入力してクライアントアプリにサインインします。
2. TOTP MFA チャレンジが呼び出され、アプリが一時パスワードを入力するプロンプトをユーザーに表示します。
3. ユーザーは、関連付けられた TOTP 生成アプリから一時パスワードを取得します。
4. ユーザーが TOTP コードをクライアントアプリに入力します。アプリは、コードを検証するよう Amazon Cognito サービスに通知します。サインインごとに、[RespondToAuthChallenge](#) を呼び出して新しい TOTP 認証チャレンジに対するレスポンスを取得する必要があります。
5. Amazon Cognito によってトークンが検証されると、サインインが成功し、ユーザーは認証フローを続行します。

TOTP トークンを削除

最後に、アプリは TOTP 設定を非アクティブ化することをユーザーに許可する必要があります。現在、ユーザーの TOTP ソフトウェアトークンを削除することはできません。ユーザーのソフトウェアトークンを置き換えるには、新しいソフトウェアトークンを関連付けて検証します。ユーザーの TOTP MFA を無効にするには、[SetUserMFAPreference](#) を呼び出して、MFA を使用しないか、SMS MFA のみを使用するようにユーザーを変更します。

1. MFA をリセットしたいユーザーのためのインターフェイスをアプリケーション内に作成します。このインターフェイスでユーザーにパスワードの入力を求めます。
2. Amazon Cognito が TOTP MFA チャレンジを返す場合は、ユーザーの MFA 設定を [SetUserMFAPreference](#) で更新します。
3. アプリで、MFA を非アクティブ化したことをユーザーに伝え、再度サインインするよう促します。

マネージドログイン TOTP MFA 用の AWS WAF ウェブ ACL の設定

ユーザープールに関連付けられた AWS WAF ウェブ ACL があり、ウェブ ACL のルールが CAPTCHA を示している場合、マネージドログイン TOTP 登録で回復不可能なエラーが発生する可能性があります。AWS WAF CAPTCHA ルールは、マネージドログインとクラシックホスト UI の TOTP MFA にのみこの効果があります。SMS MFA は影響を受けません。

CAPTCHA ルールにより、ユーザーが TOTP MFA の設定を完了できない場合、Amazon Cognito は次のエラーを表示します。

Request not allowed due to WAF captcha. (WAF captcha によりリクエストは許可されていません。)

このエラーは、がユーザープールがバックグラウンドで行う [AssociateSoftwareToken](#) および [VerifySoftwareToken](#) API リクエストに 응답して CAPTCHA AWS WAF をプロンプトすると発生します。CAPTCHA アクションを含み、マネージドログインページの TOTP に影響しないルールを作成するには、ルール内の CAPTCHA アクションから AssociateSoftwareToken と VerifySoftwareToken の x-amzn-cognito-operation-name ヘッダー値を除外します。

次のスクリーンショットは、x-amzn-cognito-operation-name ヘッダー値が AssociateSoftwareToken または VerifySoftwareToken でないすべてのリクエストに CAPTCHA アクションを適用する AWS WAF ルールの例を示しています。

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.

AWS WAF ウェブ ACLs「」を参照してください[AWS WAF ウェブ ACL をユーザープールに関連付ける](#)。Amazon Cognito

脅威保護を備えた高度なセキュリティ

ユーザープールを作成すると、Amazon Cognito コンソールのナビゲーションメニューで [脅威保護] にアクセスできます。脅威保護機能をオンにして、さまざまなリスクに応じて実行されるアクションをカスタマイズできます。また、監査モードを使用して、セキュリティ緩和策を適用せずに、検出されたリスクに関するメトリクスを収集できます。監査モードの場合、脅威保護は Amazon CloudWatch にメトリクスを発行します。Amazon Cognito が脅威保護の最初のイベントを生成すると、メトリクスを確認できます。「[脅威保護メトリクスの表示](#)」を参照してください。

脅威保護 (以前は高度なセキュリティ機能と呼ばれていました) は、ユーザープール内の望ましくないアクティビティをモニタリングするツールと、潜在的に悪意のあるアクティビティを自動的にシャットダウンする設定ツールのセットです。脅威保護には、標準認証オペレーション用とカスタム認証オペレーション用の異なる設定オプションがあります。例えば、追加のセキュリティ要素を設定済みのカスタム認証で、疑わしいサインインを使用したユーザーに通知を送信するが、基本的なユーザー名/パスワード認証を使用した同じリスクレベルのユーザーをブロックする場合などです。

脅威保護は、プラス機能プランで利用できます。詳細については、「[ユーザープールの機能プラン](#)」を参照してください。

以下のユーザープールオプションは、脅威保護のコンポーネントです。

漏えいした認証情報

ユーザーは複数のユーザーアカウントのパスワードを再利用します。Amazon Cognito の認証情報漏えい機能では、ユーザー名とパスワードの公開漏えいに関するデータを収集し、ユーザーの認証情報を漏えいした認証情報のリストと比較します。漏えいした認証情報の検出では、よく推測されるパスワードもチェックされます。ユーザープールのユーザー名とパスワードの標準認証フローで、漏えいした認証情報を確認できます。Amazon Cognito は、セキュアリモートパスワード (SRP) またはカスタム認証で漏えいした認証情報を検出しません。

漏えいした認証情報の確認を促すユーザーアクションと、それに応じて Amazon Cognito に実行してほしいアクションを選択できます。サインイン、サインアップ、パスワード変更のイベントでは、Amazon Cognito はサインインをブロックするか、サインインを許可することができます。どちらの場合も、Amazon Cognito はユーザーアクティビティログを生成し、そこでイベントに関する詳細情報を確認できます。

詳細情報

[漏えいした認証情報の検出の使用](#)

アダプティブ認証

Amazon Cognito は、ユーザーのサインインリクエストから位置情報とデバイス情報を確認し、自動応答を適用してユーザープールのユーザーアカウントを疑わしいアクティビティから保護します。ユーザーアクティビティをモニタリングし、ユーザー名パスワードと SRP、カスタム認証で検出されたりスクレレベルへの応答を自動化できます。

脅威保護を有効にすると、Amazon Cognito はユーザーアクティビティにリスクスコアを割り当てます。疑わしいアクティビティには自動応答を割り当てることができます。たとえば、MFA を義務付けたり、ログインをブロックしたり、アクティビティの詳細とリスクスコアを記録したりできます。また、疑わしいアクティビティをユーザーに通知する E メールメッセージを自動的に送信して、ユーザーがパスワードのリセットやその他の自発的なアクションを実行できるようにすることもできます。

詳細情報

[アダプティブ認証の使用](#)

IP アドレスの許可リストと拒否リスト

Amazon Cognito の脅威保護を [フル機能] モードで使用することで、IP アドレスの [常にブロック] と [常に許可] の例外を作成できます。常にブロック 例外リストの IP アドレスからのセッションは、アダプティブ認証によってリスクレベルが割り当てられておらず、ユーザープールにサインインできません。

IP アドレスの許可リストとブロックリストについて知っておくべきこと

- [常にブロック] と [常に許可] は CIDR 形式で示す必要があります。例えば、24 ビットマスクを 192.0.2.0/24、単一の IP アドレスを 192.0.2.252/32 で示します。
- [常にブロック] する IP 範囲の IP アドレスを持つデバイスは、SDK ベースやマネージドログインのアプリケーションを使用してサインアップまたはサインインできませんが、サードパーティの IdP を使用してサインインできます。
- [常に許可] と [常にブロック] のリストはトークンの更新には影響しません。
- Amazon Cognito は、[常に許可] する IP 範囲のデバイスにアダプティブ認証 MFA ルールを適用しませんが、漏えいした認証情報のルールを適用します。

ログのエクスポート

脅威保護は、ユーザープールに対するユーザーの認証リクエストの詳細なログを記録します。これらのログには、脅威評価、ユーザー情報、ロケーションやデバイスなどのセッションメ

タデータが含まれます。これらのログについて、保持と分析のための外部アーカイブを作成できます。Amazon Cognito ユーザープールは、脅威保護ログを Amazon S3、CloudWatch Logs、Amazon Data Firehose にエクスポートします。詳細については、「[ユーザーイベント履歴の表示とエクスポート](#)」を参照してください。

詳細情報

[脅威保護ユーザーアクティビティログのエクスポート](#)

トピック

- [脅威保護に関する考慮事項と制限事項](#)
- [ユーザープールで脅威保護を有効にする](#)
- [脅威保護の強制適用の概念](#)
- [標準認証とカスタム認証の脅威保護](#)
- [脅威保護の前提条件](#)
- [脅威保護の設定](#)
- [漏えいした認証情報の検出の使用](#)
- [アダプティブ認証の使用](#)
- [アプリケーションにおける脅威保護のためのデータ収集](#)

脅威保護に関する考慮事項と制限事項

脅威保護オプションは認証フローによって異なる

Amazon Cognito は、アダプティブ認証と、認証フローの USER_PASSWORD_AUTH と ADMIN_USER_PASSWORD_AUTH による漏えいした認証情報の検出の両方をサポートしています。USER_SRP_AUTH のアダプティブ認証のみを有効にできます。フェデレーティッドサインインでは脅威保護を使用できません。

常にブロックする IP はリクエストクォータにカウントされる

ユーザープールの 常にブロック 例外リストの IP アドレスからのブロックされたリクエストは、ユーザープールの [リクエストレートクォータ](#) に貢献します。

脅威保護にはレート制限は適用されない

一部の悪意のあるトラフィックは、分散型サービス拒否 (DDoS) 攻撃など、大量のリクエストの特徴を持っています。Amazon Cognito が受信トラフィックに適用するリスク評価はリクエストごとに行

われるため、リクエストボリュームは考慮されません。大量のイベント内における個々のリクエストは、大量攻撃でのロールに関連しないアプリケーションレイヤーの理由で、リスクスコアと自動レスポンスを受け取る場合があります。ユーザープールにボリュームメトリック攻撃に対する防御を実装するには、AWS WAF ウェブ ACLs を追加します。詳細については、「[AWS WAF ウェブ ACL をユーザープールに関連付ける](#)」を参照してください。

脅威保護は M2M リクエストには影響しません

クライアント認証情報の付与は、ユーザーアカウントに接続しないマシンツーマシン (M2M) 認可を目的としています。脅威保護は、ユーザープール内のユーザーアカウントとパスワードのみをモニタリングします。M2M アクティビティでセキュリティ機能を実装するには、リクエストレートとコンテンツをモニタリング AWS WAF するための の機能を検討してください。詳細については、「[AWS WAF ウェブ ACL をユーザープールに関連付ける](#)」を参照してください。

ユーザープールで脅威保護を有効にする

Amazon Cognito user pools console

ユーザープールで脅威保護を有効にするには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. まだ有効にしていない場合は、[設定]メニューからプラス機能プランを有効にします。
5. [脅威保護] メニューを選択し、[アクティブ化] を選択します。
6. [Save changes] (変更の保存) をクリックします。

API

[CreateUserPool](#) API リクエストまたは [UpdateUserPool](#) API リクエストで、機能プランをプラスに設定します。次のリクエスト本文の部分的な例では、脅威保護をフル機能モードに設定しています。完全なリクエストの例については、「[例](#)」を参照してください。

```
"UserPoolAddOns": {
  "AdvancedSecurityMode": "ENFORCED"
}
```

脅威保護は、アカウント乗っ取りの兆候がないかについてユーザーオペレーションをモニタリングし、影響を受けるユーザーアカウントのセキュリティ保護に自動的に応答する機能の集合語です。脅威保護設定は、ユーザーが標準認証フローとカスタム認証フローでサインインするときに適用できます。

脅威保護は、ユーザーのサインイン、サインアウト、その他のアクティビティを詳細に記録する[ログを生成](#)します。これらのログは、サードパーティーのシステムにエクスポートできます。詳細については、「[ユーザーイベント履歴の表示とエクスポート](#)」を参照してください。

脅威保護の強制適用の概念

脅威保護は、ユーザープールがユーザーアクティビティをモニタリングし、リスクレベルを割り当て、ログを生成する監査のみモードで開始されます。ベストプラクティスとして、フル機能モードを有効にする前に、監査のみモードで2週間以上実行します。フル機能モードには、検出された危険なアクティビティと漏えいしたパスワードに対する一連の自動応答が含まれます。監査のみモードでは、Amazon Cognito が実行している脅威評価をモニタリングできます。また、擬陽性と陰性について機能をトレーニングする[フィードバックを提供](#)することもできます。

ユーザープールレベル、および個々のアプリケーションクライアントのレベルで、すべてのアプリケーションクライアントをカバーするように脅威保護の強制適用を設定できます。アプリケーションクライアント脅威保護の設定は、ユーザープールの設定を上書きします。アプリケーションクライアントで脅威保護を設定するには、Amazon Cognito コンソールでユーザープールの [アプリケーションクライアント] メニューのアプリケーションクライアント設定に移動します。そこでは、[クライアントレベルの設定を使用] により、アプリケーションクライアント専用の強制適用を設定できます。

さらに、標準認証タイプとカスタム認証タイプに対して脅威保護を個別に設定できます。

標準認証とカスタム認証の脅威保護

脅威保護を設定する方法は、ユーザープールとアプリケーションクライアントで実行している認証のタイプによって異なります。以下の各タイプの認証には、独自の強制適用モードと自動応答を使用できます。

標準認証

[標準認証] は、ユーザー名パスワードフローとマネージドログインを使用したユーザーのサインイン、サインアウト、パスワード管理です。Amazon Cognito の脅威保護は、ユーザーがマネージドログインでサインインするときや、以下の API AuthFlow パラメータを使用するときに、オペレーションをモニタリングしてリスクの兆候がないか確認します。

[InitiateAuth](#)

USER_PASSWORD_AUTH、USER_SRP_AUTH。漏えいした認証情報機能は、USER_SRP_AUTH サインイン時のパスワードにアクセスできず、このフローを用いてイベントのモニタリングや、イベントに対するアクションを行いません。

[AdminInitiateAuth](#)

ADMIN_USER_PASSWORD_AUTH、USER_SRP_AUTH。漏えいした認証情報機能は、USER_SRP_AUTH サインイン時のパスワードにアクセスできず、このフローを用いてイベントのモニタリングや、イベントに対するアクションを行いません。

標準認証の [強制適用モード] を [監査のみ] または [フル機能] に設定できます。標準認証の脅威モニタリングを無効にするには、脅威保護を [強制適用なし] に設定します。

カスタム認証

カスタム認証は、[カスタムチャレンジ Lambda トリガー](#)によるユーザーサインインです。マネージドログインではカスタム認証を実行できません。Amazon Cognito 脅威保護は、InitiateAuth および AdminInitiateAuth の API AuthFlow パラメータ CUSTOM_AUTH を使用してサインインするときに、リスクの指標についてオペレーションをモニタリングします。

カスタム認証の [強制適用モード] は、[監査のみ]、[フル機能]、または [強制適用なし] に設定できます。[強制適用なし] オプションは、脅威保護の他の機能に影響を与えることなく、カスタム認証の脅威モニタリングを無効にします。

脅威保護の前提条件

開始するには、以下が必要です。

- アプリクライアントを持つユーザープール。詳細については、「[ユーザープールの開始方法](#)」を参照してください。
- Amazon Cognito コンソールで多要素認証 (MFA) を [Optional] (オプション) に設定して、リスクに基づくアダプティブ認証機能を使用する。詳細については、「[ユーザープールに MFA を追加します](#)」を参照してください。
- 通知に E メールを使用している場合は、[Amazon SES コンソール](#)に移動して、通知 E メールで使用する E メールアドレスまたはドメインの設定と検証を実行します。Amazon SES の詳細については、「[Amazon SES での ID の検証](#)」を参照してください。

脅威保護の設定

ユーザープールの脅威保護を設定するには、次の手順に従います。

Note

Amazon Cognito ユーザープールコンソールでアプリケーションクライアント別の脅威保護設定をセットアップするには、[アプリケーションクライアント] メニューでアプリケーションクライアントを選択し、[クライアントレベルの設定を使用] を選択します。

AWS マネジメントコンソール

ユーザープールの脅威保護を設定するには

1. [Amazon Cognito コンソール](#)に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [脅威保護] メニューを選択し、[アクティブ化] を選択します。
5. 設定する脅威保護方法を、標準認証とカスタム認証から選択します。カスタム認証と標準認証にはさまざまな強制適用モードを設定できますが、それらはフル機能モードでの自動応答の設定を共有します。
6. [Edit] (編集) を選択します。
7. [強制適用モード] を選択します。検出されたリスクへの対応をすぐに開始するには、[フル機能] を選択し、漏えいした認証情報とアダプティブ認証への自動レスポンスを設定します。ユーザーレベルのログと CloudWatch で情報を収集するには、[監査のみ] を選択します。

アクションを有効にする前の 2 週間、脅威保護を監査モードにしておくことをお勧めします。この間に、Amazon Cognito は、アプリケーションユーザーの使用パターンを学習でき、イベントフィードバックを提供してレスポンスを調整できます。

8. [Audit only] (監査のみ) を選択した場合、[Save changes] (変更の保存) を選択してください。[Full function] (完全な機能) を選択した場合:
 - a. 侵害された認証情報への応答は、[Custom] (カスタム) アクションの実行または使用、または [Cognito defaults] (Cognito デフォルト) を選択します。Cognito デフォルトでは、

- i. サインイン、サインアップ、およびパスワードの変更時に、侵害された認証情報を検出します。
- ii. 侵害された認証情報は、[Block sign-in] (サインインをブロックする) で応答します。
- b. [漏えいした認証情報] で [カスタム] アクションを選択した場合、Amazon Cognito で [イベントの検出] に使用するユーザープールアクションと、Amazon Cognito で実行する [漏えいした認証情報に関する応答] を選択します。侵害された認証情報によるサインインをブロックまたはサインインを許可することができます。
- c. [Adaptive authentication] (アダプティブ認証) で、悪意のあるサインイン試行への応答方法を選択します。[Custom] (カスタム) アクションを実行または使用、または悪意のあるアクティビティの疑いに対応するために [Cognito defaults] (Cognito デフォルト) を選択します。[Cognito defaults] (Cognito デフォルト) を選択した場合、Amazon Cognito はすべてのリスクレベルでサインインをブロックし、ユーザーに通知しません。
- d. [Adaptive authentication] (アダプティブ認証) で、[Custom] (カスタム) アクションを選択した場合、重要度レベルに基づいて検出されたリスクに対して、Amazon Cognito で実行する [Automatic risk response] (自動リスク対応) アクションを選択します。リスクのレベルに応じて対応を割り当てる場合、より高いレベルのリスクに対して、より制限の少ない対応を割り当てることはできません。リスクレベルには、次の対応を割り当てるすることができます。
 - i. サインインを許可する - 予防策をとりません。
 - ii. MFA のオプション - ユーザーが MFA を設定している場合、Amazon Cognito はサインイン時に常に SMS またはタイムベースドワンタイムパスワード (TOTP) の追加要素を提供するようユーザーに要求します。ユーザーに MFA が設定されていない場合は、通常どおりサインインを続行できます。
 - iii. MFA を要求 - ユーザーが MFA を設定している場合、Amazon Cognito はサインイン時に常に SMS または TOTP ファクターの追加要素の提供を要求します。ユーザーに MFA が設定されていない場合、Amazon Cognito は MFA を設定するよう促します。ユーザーに MFA を自動要求する前に、SMS MFA の電話番号をキャプチャするメカニズム、または TOTP MFA の認証アプリケーションを登録するメカニズムを、お客様のアプリケーションに設定してください。
 - iv. サインインをブロックする - ユーザーがサインインできないようにします。
 - v. ユーザーに通知する - Amazon Cognito が検出したリスクと応答に関する情報を E メールでユーザーに送信します。送信するメッセージの E メールメッセージテンプレートをカスタマイズできます。

9. 前の手順で [Notify user] (ユーザーに通知する) を選択した場合、アダプティブ認証に使用する E メール配信設定と E メールメッセージテンプレートをカスタマイズできます。
- a. [E メール設定] で、アダプティブ認証に使用する [SES リージョン]、[送信元の E メールアドレス]、[送信者の名前]、[返信先 E メールアドレス] を選択します。ユーザープールの E メールメッセージを Amazon Simple Email Service の統合方法の詳細については、「[Amazon Cognito ユーザープールに使用する Eメールの設定](#)」を参照してください。

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](#) Email message - HTML [Reset to default](#)
We observed an unrecognized sign-in to your <!DOCTYPE html>

- b. [Email templates] (E メールテンプレート) を展開して、E メールメッセージの HTML とプレーンテキストバージョンの両方でアダプティブ認証通知をカスタマイズします。E メールメッセージテンプレートの詳細については、「[メッセージテンプレート](#)」を参照してください。

10. [IP アドレスの例外] を展開して、脅威保護のリスク評価に関係なく、常に許可またはブロックする IPv4 または IPv6 アドレス範囲の [常に許可] または [常にブロック] のリストを作成します。 [CIDR 表記](#) で IP アドレスの範囲を指定します (例: 192.168.100.0/24)。
11. [Save changes] (変更の保存) をクリックします。

API (user pool)

ユーザープールの脅威保護を設定するには、UserPoolId パラメータを含むが、ClientId パラメータを含まない [SetRiskConfiguration](#) API リクエストを送信します。ユーザープール用のリクエスト本文の例を次に示します。このリスク設定は、リスクの重大度に基づいて一連のアクションをエスカレートし、すべてのリスクレベルでユーザーに通知します。これは、漏えいした認証情報ブロックをサインアップオペレーションに適用します。

この設定を適用するには、別の [CreateUserPool](#) API リクエストまたは [UpdateUserPool](#) API リクエストの ENFORCED に AdvancedSecurityMode を設定する必要があります。この例における {username} のようなプレースホルダーテンプレートの詳細については、「[MFA、認証、検証、招待メッセージの設定](#)」を参照してください。

```
{
  "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)

アプリケーションクライアントの脅威保護を設定するには、UserPoolId パラメータと ClientId パラメータを含む [SetRiskConfiguration](#) API リクエストを送信します。次に、アプリケーションクライアント用のリクエストの本文の例を示します。このリスク設定は、ユーザープール設定よりも重大であり、高リスクのエントリをブロックします。また、漏えいした認証情報ブロックを適用して、サインアップ、サインイン、パスワードリセットの各オペレーションも行います。

この設定を適用するには、別の [CreateUserPool](#) API リクエストまたは [UpdateUserPool](#) API リクエストの ENFORCED に AdvancedSecurityMode を設定する必要があります。この例における {username} のようなプレースホルダーテンプレートの詳細については、「[MFA、認証、検証、招待メッセージの設定](#)」を参照してください。

```
{
```

```

"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"
}
```

漏えいした認証情報の検出の使用

Amazon Cognito は、ユーザーのユーザー名とパスワードが他の場所で侵害されたかどうかを検出できます。これは、ユーザーが複数のサイトで認証情報を再利用したり、安全でないパスワードを使用したりするときに発生します。Amazon Cognito は、マネージドログインと Amazon Cognito API で、ユーザー名とパスワードでサインインする[ローカルユーザー](#)をチェックします。

Amazon Cognito コンソールの [脅威保護] メニューで、[漏えいした認証情報] を設定できます。[Event detection] (イベント検出) を設定して、侵害された認証情報を監視するユーザーイベントを選択します。[Compromised credentials responses] (侵害された認証情報の応答) を設定し、侵害された認証情報が検出された場合にユーザーを許可するかブロックするかを選択します。Amazon Cognito は、サインイン時、サインアップ時、パスワード変更時に侵害された認証情報をチェックすることができます。

[Allow sign-in] (サインインを許可する) を選択する場合、Amazon CloudWatch Logs を確認して、Amazon Cognito がユーザーイベントに対して行う評価を監視できます。詳細については、「[脅威保護メトリクスの表示](#)」を参照してください。[Block sign-in] (サインインをブロックする) を選択する場合、Amazon Cognito は、侵害された認証情報を使用するユーザーによるサインインを防止します。Amazon Cognito がユーザーのサインインをブロックすると、ユーザーの [UserStatus](#) が RESET_REQUIRED に設定されます。RESET_REQUIRED ステータスのユーザーは、再度サインインする前にパスワードを変更する必要があります。

漏えいした認証情報では、次のユーザーアクティビティのパスワードをチェックできます。

サインアップ

ユーザープールは、ユーザーが [SignUp](#) オペレーションおよびマネージドログインのサインアップページで送信したパスワードをチェックし、漏えいの兆候を確認します。

サインイン

ユーザープールは、ユーザーがパスワードベースのサインインで送信したパスワードをチェックし、漏えいの兆候を確認します。Amazon Cognito は、[AdminInitiateAuth](#) の

ADMIN_USER_PASSWORD_AUTH フロー、[InitiateAuth](#) の USER_PASSWORD_AUTH フロー、および両方の USER_AUTH フローの PASSWORD オプションを確認できます。

現在 Amazon Cognito では、Secure Remote Password (SRP) フローでのサインイン操作に対する侵害された認証情報のチェックが行われません。SRP はサインイン時にハッシュ化されたパスワード証明書を送信します。Amazon Cognito は内部でパスワードにアクセスできないため、クライアントがプレーンテキストで渡したパスワードのみを評価できます。

パスワードのリセット

ユーザープールは、セルフサービスのパスワードリセットオペレーション [ConfirmForgotPassword](#) を使用して、新規ユーザーのパスワードを設定するオペレーションにおける漏えいの兆候をチェックします。このオペレーションに必要なコードは、[ForgotPassword](#) と [AdminResetUserPassword](#) によって生成されます。

漏えいした認証情報では、[AdminSetUserPassword](#) で設定された仮パスワードや永続的な管理者設定パスワードをチェックしません。ただし、仮パスワードの場合、ユーザープールは [RespondToAuthChallenge](#) と [AdminRespondToAuthChallenge](#) で NEW_PASSWORD_REQUIRED チャレンジへのレスポンスのパスワードをチェックします。

漏洩した認証情報の保護をユーザープールに追加するには、「[脅威保護を備えた高度なセキュリティ](#)」を参照してください。

アダプティブ認証の使用

アダプティブ認証では、リスクレベルの上昇に対応して、疑わしいサインインをブロックする、または 2 番目の要素認証を追加するようにユーザープールを設定できます。Amazon Cognito はサインインの試行ごとに、サインインリクエストが侵害されたソースからのものである可能性についてリスクスコアを生成します。このリスクスコアは、アプリケーションが提供するデバイスとユーザーの要因、および Amazon Cognito がリクエストから派生するその他の要因に基づいています。Amazon Cognito によるリスク評価に影響する要素には、IP アドレスとユーザーエージェント、ユーザーエージェント、および他のサインイン試行からの地理的な距離があります。アダプティブ認証は、Amazon Cognito がユーザーのセッションでリスクを検出し、ユーザーがまだ MFA メソッドを選択していない場合に、ユーザープール内のユーザーに対して多要素認証 (MFA) を有効にする、または多要素認証 (MFA) を要求することができます。ユーザーに MFA を有効にすると、アダプティブ認証の設定方法にかかわらず、認証中に必ず 2 つ目の要素を提供または設定するように求めるチャレンジが常に表示されます。ユーザーの観点から見ると、アプリは MFA の設定を支援し、また、オプションとして、Amazon Cognito では、ユーザーが追加の要素を設定するまで、ユーザーが再度サインインできないようにすることもできます。

Amazon Cognito は、サインイン試行、試行のリスクレベル、および失敗したチャレンジに関するメトリクスを Amazon CloudWatch にパブリッシュします。詳細については、「[脅威保護メトリクスの表示](#)」を参照してください。

アダプティブ認証をユーザープールに追加するには、「[脅威保護を備えた高度なセキュリティ](#)」を参照してください。

トピック

- [アダプティブ認証の概要](#)
- [API リクエストへのユーザーデバイスおよびセッションデータの追加](#)
- [ユーザーイベント履歴の表示とエクスポート](#)
- [イベントフィードバックを提供します](#)
- [通知メッセージの送信](#)

アダプティブ認証の概要

Amazon Cognito コンソールの [脅威保護] メニューで、アダプティブ認証の設定を選択できます。これには、リスクレベル別に実行するアクション、ユーザーへの通知メッセージのカスタマイズなどが含まれます。すべてのアプリケーションクライアントにグローバルな脅威保護設定を割り当てる一方、個々のアプリケーションクライアントにはクライアントレベルの設定を適用できます。

Amazon Cognito アダプティブ認証は、各ユーザーセッションに「高」、「中」、「低」、「リスクなし」のいずれかのリスクレベルを割り当てます。

[Enforcement method] (強制実行メソッド) を [Audit-only] (監査専用) から [Full-function] (完全な機能) に変更する場合は、選択肢を慎重に検討してください。リスクレベルに適用する自動応答は、Amazon Cognito が同じ特性を持つ後続のユーザーセッションに割り当てるリスクレベルに影響します。例えば、Amazon Cognito が最初にリスクが高いと評価したユーザーセッションに何もしないか、[Allow] (許可) を選択すると、Amazon Cognito では同様のセッションのリスクが低いと見なします。

リスクレベルごとに、以下のオプションから選択できます。

オプション	アクション
許可	ユーザーは、追加要素なしでサインインできます。

オプション	アクション
オプションの MFA	第 2 要素を設定しているユーザーは、サインインするために第 2 要素のチャレンジを完了する必要があります。SMS の電話番号と TOTP ソフトウェアトークンが利用可能な第 2 要素です。2 番目の要素が設定されていないユーザーは、1 つの認証情報のみでサインインできます。
MFA が必要	第 2 要素を設定しているユーザーは、サインインするために第 2 要素のチャレンジを完了する必要があります。Amazon Cognito は、第 2 要素が設定されていないユーザーのサインインをブロックします。
ブロック	Amazon Cognito は、指定されたリスクレベルですべてのサインイン試行をブロックします。

 Note

SMS を 2 番目の認証要素として使用するために電話番号を検証する必要はありません。

API リクエストへのユーザーデバイスおよびセッションデータの追加

API を使用してユーザーのサインアップ、サインイン、パスワードのリセットを行う際に、ユーザーのセッションに関する情報を収集し、Amazon Cognito の脅威保護に渡すことができます。この情報には、ユーザーの IP アドレスと一意のデバイス識別子が含まれます。

ユーザーと Amazon Cognito の間に、プロキシサービスやアプリケーションサーバーなど、中間ネットワークデバイスがある場合があります。ユーザーのコンテキストデータを収集して Amazon Cognito に渡すことで、アダプティブ認証がサーバーやプロキシではなく、ユーザーエンドポイントの特性に基づいてリスクを計算できます。クライアント側アプリが Amazon Cognito API オペレーションを直接呼び出す場合、アダプティブ認証は送信元 IP アドレスを自動的に記録します。ただし、デバイスのフィンガープリントも収集しない限り、user-agent などの他のデバイス情報は記録されません。

このデータを Amazon Cognito コンテキストデータ収集ライブラリで生成して、Amazon Cognito の脅威保護に送信するには、[ContextData](#) パラメータと [UserContextData](#) パラメータを使用します。コンテキストデータ収集ライブラリは AWS SDKs に含まれています。詳細については、「[Amazon Cognito の認証と認可を、ウェブアプリケーションとモバイルアプリケーションに統合する](#)」を参照してください。プラス機能プランを使用している場合は、ContextData を送信できます。詳細については、「[脅威保護の設定](#)」を参照してください。

アプリケーションサーバーから以下の Amazon Cognito 認証 API オペレーションを呼び出す場合、ContextData パラメータにユーザーのデバイスの IP を渡します。また、サーバー名、サーバーパス、およびエンコードされたデバイスフィンガープリントデータを渡します。

- [AdminInitiateAuth](#)
- [AdminRespondToAuthChallenge](#)

Amazon Cognito の認証されていない API オペレーションを呼び出すと、Amazon Cognito の脅威保護に UserContextData を送信できます。このデータには、EncodedData パラメータのデバイスフィンガープリントが含まれます。また、次の条件を満たす場合、UserContextData に IPAddress パラメータを送信できます。

- ユーザープールをプラス機能プランで使用しています。詳細については、「[ユーザープールの機能プラン](#)」を参照してください。
- アプリクライアントにはクライアントシークレットがあります。詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。
- アプリクライアントで [Accept additional user context data] (追加のユーザーコンテキストデータを受け入れる) をアクティブ化しました。詳細については、「[追加のユーザーコンテキストデータの受け入れ \(AWS マネジメントコンソール\)](#)」を参照してください。

アプリは、以下の Amazon Cognito の認証されていない API オペレーションで、UserContextData パラメータにエンコードされたデバイスフィンガープリントデータとユーザーのデバイスの IP アドレスを入力できます。

- [InitiateAuth](#)
- [RespondToAuthChallenge](#)
- [SignUp](#)
- [ConfirmSignUp](#)
- [ForgotPassword](#)

- [ConfirmForgotPassword](#)
- [ResendConfirmationCode](#)

追加のユーザーコンテキストデータの受け入れ (AWS マネジメントコンソール)

ユーザープールは [Accept additional user context data] (追加のユーザーコンテキストデータを受け入れる) 機能をアクティブ化すると、UserContextData パラメータで IP アドレスを受け入れます。次の場合は、この機能をアクティブ化する必要はありません。

- ユーザーは、[AdminInitiateAuth](#) のような認証された API オペレーションでのみサインインし、ContextData パラメータを使用します。
- 認証されていない API オペレーションから、IP アドレスではなく、デバイスのフィンガープリントのみを Amazon Cognito の脅威保護に送信します。

Amazon Cognito コンソールでアプリケーションクライアントを次のように更新し、追加のユーザーコンテキストデータのサポートを追加します。

1. [Amazon Cognito コンソール](#)にサインインします。
2. ナビゲーションペインで [ユーザープールの管理] を選択してから、編集するユーザープールを選択します。
3. [アプリケーションクライアント] メニューを選択します。
4. アプリケーションクライアントを選択または作成します。詳細については、「[ユーザープールのアプリケーションクライアントの設定](#)」を参照してください。
5. [App client information] (アプリのクライアント情報) コンテナから [Edit] (編集) を選択します。
6. アプリケーションクライアントの [Advanced authentication settings] (詳細な認証設定) で、[Accept additional user context data] (追加のユーザーコンテキストデータを受け入れる) を選択します。
7. [Save changes] (変更の保存) をクリックします。

Amazon Cognito API でユーザーコンテキストデータを受け入れるようにアプリケーションクライアントを設定するには、[CreateUserPoolClient](#) または [UpdateUserPoolClient](#) リクエストで EnablePropagateAdditionalUserContextData を true に設定します。ウェブまたはモバイルアプリで脅威保護を使用する方法については、「[アプリケーションにおける脅威保護のためのデータ収集](#)」を参照してください。アプリがサーバーから Amazon Cognito を呼び出すときに、クライアント側からユーザーコンテキストデータを収集します。以下は、JavaScript SDK メソッド getData 使用する例です。

```
var EncodedData =  
    AmazonCognitoAdvancedSecurityData.getData(username, userPoolId, clientId);
```

アダプティブ認証を使用するようにアプリを設計する場合は、最新の Amazon Cognito SDK をアプリに組み込むことをお勧めします。SDK の最新バージョンでは、デバイス ID、モデル、およびタイムゾーンなどのデバイスフィンガープリント情報を収集します。Amazon Cognito SDK の詳細については、「[ユーザープール SDK のインストール](#)」を参照してください。Amazon Cognito の脅威保護では、アプリケーションが正しい形式で送信したイベントのみを保存し、リスクスコアを割り当てます。Amazon Cognito がエラーレスポンスを返す場合は、リクエストに有効なシークレットハッシュが含まれていることと、IPAddress パラメータが、有効な IPv4 または IPv6 アドレスであることを確認します。

ContextData および UserContextData リソース

- AWS Amplify Android 用 SDK: [GetUserContextData](#)
- AWS Amplify SDK for iOS: [userContextData](#)
- JavaScript: [amazon-cognito-advanced-security-data.min.js](#)

ユーザーイベント履歴の表示とエクスポート

脅威保護を有効にすると、Amazon Cognito は、ユーザーによる認証イベントごとにログを生成します。デフォルトでは、Amazon Cognito コンソールの [ユーザー] メニューまたは [AdminListUserAuthEvents](#) API オペレーションを使用して、ユーザーログを表示できます。また、これらのイベントを、CloudWatch Logs、Amazon S3、Amazon Data Firehose などの外部システムにエクスポートすることもできます。エクスポート機能を使用すると、アプリケーション内のユーザーアクティビティに関するセキュリティ情報について、独自のセキュリティ分析システムにアクセスさせることが容易になります。

トピック

- [ユーザーイベント履歴の表示 \(AWS マネジメントコンソール\)](#)
- [ユーザーイベント履歴の表示 \(API/CLI\)](#)
- [ユーザー認証イベントのエクスポート](#)

ユーザーイベント履歴の表示 (AWS マネジメントコンソール)

ユーザーのサインイン履歴を表示するには、Amazon Cognito コンソールの [ユーザー] メニューでユーザーを選択できます。Amazon Cognito は、ユーザーイベント履歴を 2 年間保持します。



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	Chrome Mobile, Android Mobile	Seattle	-
Jan 18, 2018 9:20:28 PM	Sign In	In Progress	High	Account Takeover	Password:Success	67.132.130.174	Chrome Mobile, Android Mobile	Seattle	-
Jan 18, 2018 9:18:18 PM	Sign In	Pass	-	No Risk	Password:Success	67.132.130.174	Chrome Mobile, Android Mobile	Seattle	Invalid

per page
 < 1 2 3 >

各サインインイベントにはイベント ID があります。イベントには、場所、デバイスの詳細、およびリスク検出結果など、対応するコンテキストデータもあります。

また、イベント ID を Amazon Cognito がイベントを記録した時点で発行したトークンと関連付けることができます。ID とアクセストークンには、ペイロードにこのイベント ID が含まれます。Amazon Cognito はまた、更新トークンの使用を元のイベント ID に関連付けます。元のイベント ID は、Amazon Cognito トークンの発行につながったサインインイベントのイベント ID まで追跡できます。システム内のトークンの使用は、特定の認証イベントまで追跡できます。詳細については、「[ユーザープール JSON ウェブトークン \(JWT\) の理解](#)」を参照してください。

ユーザーイベント履歴の表示 (API/CLI)

Amazon Cognito API オペレーション [AdminListUserAuthEvents](#) を使用して、または [admin-list-user-auth-events](#) を使用して AWS Command Line Interface (AWS CLI) を使用して、ユーザーイベント履歴をクエリできます。

AdminListUserAuthEvents request

次の AdminListUserAuthEvents 用のリクエスト本文は、1 人のユーザーの最新のアクティビティログを返します。

```
{
  "UserPoolId": "us-west-2_EXAMPLE",
  "Username": "myexampleuser",
  "MaxResults": 1
}
```

```
}
```

admin-list-user-auth-events request

次の admin-list-user-auth-events 用のリクエストは、1 人のユーザーの最新のアクティビティログを返します。

```
aws cognito-idp admin-list-user-auth-events --max-results 1 --username myexampleuser  
--user-pool-id us-west-2_EXAMPLE
```

Response

Amazon Cognito は、両方のリクエストに同じ JSON レスポンス本文を返します。以下は、リスク要因が含まれていないことが検出されたマネージドログインのサインインイベントのレスポンス例です。

```
{  
  "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]"
}
```

ユーザー認証イベントのエクスポート

脅威保護から外部システムにユーザーイベントをエクスポートするようにユーザープールを設定します。サポートされている外部システムである Amazon S3、CloudWatch Logs、Amazon Data Firehose は、送信または取得するデータの AWS 請求にコストを追加する可能性があります。詳細については、「[脅威保護ユーザーアクティビティログのエクスポート](#)」を参照してください。

AWS マネジメントコンソール

1. [Amazon Cognito コンソール](#)にサインインします。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [ログストリーミング] メニューを選択します。[Edit] (編集) を選択します。
5. [ログ記録ステータス] で、[ユーザーアクティビティログのエクスポートをアクティブ化] の横にあるチェックボックスをオンにします。
6. [ログ記録先] で、ログを処理する AWS のサービスを [CloudWatch ロググループ]、[Amazon Data Firehose ストリーム]、または [S3 バケット] の中から選択します。
7. 選択すると、対応するリソースタイプがリソースセレクトターに入力されます。リストからロググループ、ストリーム、またはバケットを選択します。[作成] ボタンを選択して、選択したサービスの AWS マネジメントコンソール に移動し、新しいリソースを作成することもできます。
8. [変更を保存] を選択します。

API

ユーザーアクティビティログの送信先のタイプを 1 つ選択します。

以下は、Firehose ストリームをログの送信先として設定する SetLogDeliveryConfiguration リクエスト本文の例です。

```
{
  "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"
}
```

以下は、Amazon S3 バケットをログの送信先として設定する SetLogDeliveryConfiguration リクエスト本文の例です。

```
{
  "LogConfigurations": [
    {
      "EventSource": "userAuthEvents",
      "S3Configuration": {
        "BucketArn": "arn:aws:s3:::amzn-s3-demo-logging-bucket"
      },
      "LogLevel": "INFO"
    }
  ],
  "UserPoolId": "us-west-2_EXAMPLE"
}
```

以下は、CloudWatch ロググループをログの送信先として設定する SetLogDeliveryConfiguration リクエスト本文の例です。

```
{
  "LogConfigurations": [
    {
      "EventSource": "userAuthEvents",
      "CloudWatchLogsConfiguration": {
        "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:D0C-
EXAMPLE-LOG-GROUP"
      },
      "LogLevel": "INFO"
    }
  ],
  "UserPoolId": "us-west-2_EXAMPLE"
}
```

イベントフィードバックを提供します

イベントフィードバックは、リアルタイムでリスクの評価に反映され、リスク評価アルゴリズムを経時的に向上させます。ユーザーは、Amazon Cognito コンソールおよび API を使用して、サインイン試行の妥当性に関するフィードバックを提供できます。

Note

イベントフィードバックは、Amazon Cognito が同じ特性を持つ後続のユーザーセッションに割り当てるリスクレベルに影響します。

Amazon Cognito コンソールで、[ユーザー] メニューからユーザーを選択し、[イベントフィードバックを提供] を選択します。イベントの詳細を確認して、[Set as valid] (有効として設定) または [Set as invalid] (無効として設定) できます。

コンソールでは、[ユーザー] メニューのユーザー詳細にサインイン履歴が一覧表示されます。エントリを選択すると、イベントを有効または無効としてマークできます。また、ユーザープール API オペレーション [AdminUpdateAuthEventFeedback](#) および AWS CLI コマンド [admin-update-auth-event-feedback](#) を通じてフィードバックを提供することもできます。

Amazon Cognito コンソールで [Set as valid] (有効として設定) を選択するか、API で `valid` の `FeedbackValue` 値を指定するとき、Amazon Cognito がある程度のリスクを評価したユーザーセッションを信頼することを Amazon Cognito に伝えます。Amazon Cognito コンソールで [Set as invalid] (無効として設定) を選択するか、API で `invalid` の `FeedbackValue` 値を指定するとき、ユーザーセッションを信頼しないこと、または Amazon Cognito が十分に高いリスクレベルを評価したとは考えないことを Amazon Cognito に伝えます。

通知メッセージの送信

脅威保護により、Amazon Cognito はユーザーにリスクの高いサインイン試行を通知できます。Amazon Cognito では、サインインが有効か無効かを示すリンクを選択するようユーザーに促すこともできます。Amazon Cognito はこのフィードバックを使用して、ユーザープールのリスク検出精度を向上させます。

Note

Amazon Cognito は、ユーザーのアクションによって自動リスク対応 (サインインのブロック、サインインの許可、MFA をオプションに設定、または MFA を必須に設定) が生成された場合にのみ、ユーザーに通知メッセージを送信します。リクエストによっては、リスク

レベルが割り当てられていても、アダプティブ認証の自動リスク対応が生成されない場合があります。このようなリクエストの場合、ユーザープールは通知を送信しません。例えば、誤ったパスワードがリスク評価とともにログに記録される場合がありますが、Amazon Cognito による対応では、アダプティブ認証ルールを適用せずに、サインインを失敗させます。

[Automatic risk response] (自動リスク対応) セクションで、低、中、高のリスクケースに応じて [Notify Users] (ユーザーに通知) を選択します。

Automatic risk response Info					
Risk level	Allow sign-in	Optional MFA	Require MFA	Block sign-in	Notify user
Low risk	☒	☐	☐	☐	☒
Medium risk	☐	☒	☐	☐	☒
High risk	☐	☐	☒	☐	☒

Amazon Cognito は、ユーザーが E メールアドレスを検証したかどうかに関係なく、E メール通知をユーザーに送信します。

通知 E メールメッセージをカスタマイズして、これらのメッセージのプレーンテキストと HTML の両バージョンを提供できます。E メール通知をカスタマイズするには、脅威保護設定の [アダプティブ認証メッセージ] から [E メールテンプレート] を開きます。E メールテンプレートの詳細については、「[メッセージテンプレート](#)」を参照してください。

アプリケーションにおける脅威保護のためのデータ収集

Amazon Cognito [アダプティブ認証](#) は、ユーザーのサインイン試行のコンテキスト詳細からアカウント乗っ取りを試みた場合のリスクレベルを評価します。Amazon Cognito の脅威保護がリスクをより正確に評価できるように、アプリケーションは API リクエストにコンテキストデータを追加する必要があります。コンテキストデータは、ユーザーがユーザープールにどのように接続したかに関するコンテキスト情報を提供する、IP アドレス、ブラウザエージェント、デバイス情報、リクエストヘッダーなどの情報です。

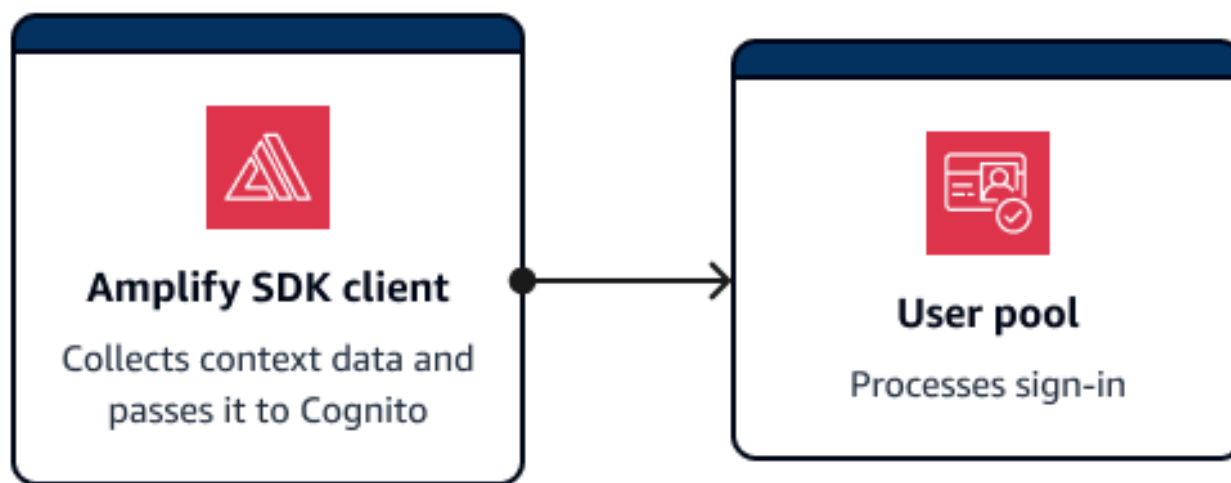
このコンテキストを Amazon Cognito に送信するアプリケーションの主な責任は、ユーザープールへの認証リクエストの EncodedData パラメータです。このデータをリクエストに追加するには、この情報を自動的に生成する SDK を使用して Amazon Cognito を実装することも、このデータを収集

する JavaScript、iOS、または Android 用のモジュールを実装することもできます。Amazon Cognito に直接リクエストを行うクライアント専用アプリケーションは AWS Amplify SDKs を実装する必要があります。中間サーバーまたは API コンポーネントを持つクライアントサーバーアプリケーションは、別の SDK モジュールを実装する必要があります。

次のシナリオでは、認証フロントエンドが、追加の設定なしでユーザーコンテキストのデータ収集を管理します。

- マネージドログインは、コンテキストデータを自動的に収集して脅威保護に送信します。
- すべての AWS Amplify ライブラリには、認証方法に組み込まれたコンテキストデータ収集があります。

Amplify を使用したクライアント専用アプリケーションでのユーザーコンテキストデータの送信



Amplify SDK は、Amazon Cognito で直接認証するモバイルクライアントをサポートします。この種類のクライアントは、Amazon Cognito パブリック API オペレーションに直接 API リクエストを行います。Amplify クライアントは、デフォルトで脅威保護のためのコンテキストデータを自動的に収集します。

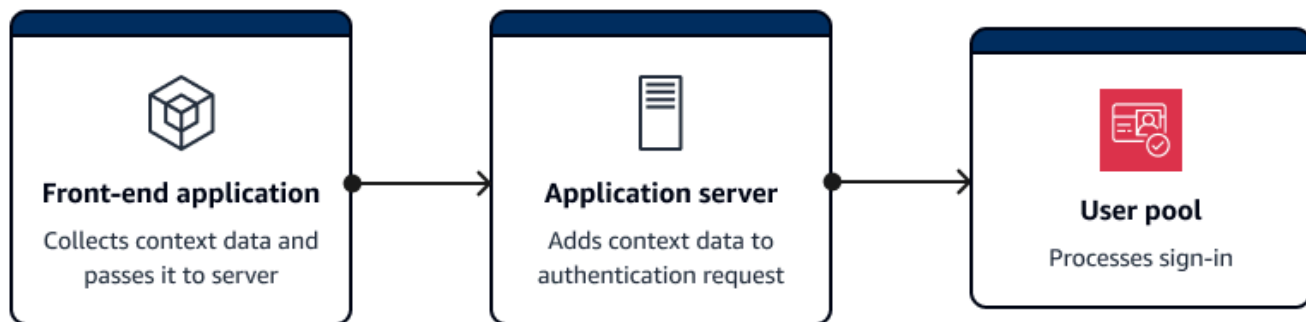
JavaScript を使用した Amplify アプリケーションは例外です。ユーザーコンテキストデータを収集する [JavaScript モジュール](#) を追加する必要があります。

通常、この設定のアプリケーションは、[InitiateAuth](#) や [RespondToAuthChallenge](#) などの認証されていない API オペレーションを使用します。[UserContextData](#) オブジェクトは、これらのオペレー

シヨンのリスクをより正確に評価するのに役立ちます。Amplify SDK は、デバイスとセッション情報を `UserContextData` の `EncodedData` パラメータに追加します。

クライアントサーバーアプリケーションでのコンテキストデータの収集

一部のアプリケーションには、ユーザー認証データを収集するフロントエンド層と、Amazon Cognito に認証リクエストを送信するアプリケーションバックエンド層があります。これは、マイクロサービスベースのウェブサーバーとアプリケーションの一般的なアーキテクチャです。これらのアプリケーションでは、パブリックコンテキストデータ収集ライブラリをインポートする必要があります。



通常、この設定のアプリケーションサーバーは、[AdminInitiateAuth](#) や [AdminRespondToAuthChallenge](#) などの認証された API オペレーションを使用します。[ContextData](#) オブジェクトは、Amazon Cognito がこれらのオペレーションのリスクをより正確に評価するのに役立ちます。`ContextData` のコンテンツは、フロントエンドがサーバーに渡したエンコードされたデータと、ユーザーによる HTTP リクエストからサーバーへの追加の詳細です。HTTP ヘッダーや IP アドレスなど、これらの追加のコンテキストの詳細により、アプリケーションサーバーにユーザーの環境の特性が提供されます。

また、アプリケーションサーバーは、[InitiateAuth](#) や [RespondToAuthChallenge](#) などの認証されていない API オペレーションでサインインする場合もあります。[UserContextData](#) オブジェクトは、これらのオペレーションにおける脅威保護のセキュリティリスク分析に役立ちます。利用可能なパブリックコンテキストデータ収集ライブラリ内のオペレーションは、認証リクエストの `EncodedData` パラメータにセキュリティ情報を追加します。さらに、追加のコンテキストデータを受け入れるようにユーザープールを設定し、ユーザーのソース IP を `UserContextData` の `IpAddress` パラメータに追加します。

クライアントサーバーアプリケーションにコンテキストデータを追加するには

1. フロントエンドアプリケーションで、[iOS、Android、または JavaScript モジュール](#)を使用してクライアントからエンコードされたコンテキストデータを収集します。
2. エンコードされたデータと認証リクエストの詳細をアプリケーションサーバーに渡します。
3. アプリケーションサーバーで、HTTP リクエストから、ユーザーの IP アドレス、関連する HTTP ヘッダー、リクエストされたサーバー名、リクエストされたパスを抽出します。これらの値を Amazon Cognito への API リクエストの [ContextData](#) パラメータに入力します。
4. SDK モジュールが収集したエンコードされたデバイスデータを使用して、API リクエストの ContextData の EncodedData パラメータを入力します。このコンテキストデータを認証リクエストに追加します。

クライアントサーバーアプリケーション用のコンテキストデータライブラリ

JavaScript

amazon-cognito-advanced-security-data.min.js モジュールは、アプリケーションサーバーに渡すことができる EncodedData を収集します。

amazon-cognito-advanced-security-data.min.js モジュールを JavaScript 設定に追加します。を、us-east-1、us-east-2、eu-west-2、または us-west-2 eu-west-1 AWS リージョン のリストから <region>に置き換えますeu-central-1。

```
<script src="https://amazon-cognito-assets.<region>.amazoncognito.com/amazon-cognito-advanced-security-data.min.js"></script>
```

EncodedData パラメータで利用できる encodedContextData オブジェクトを生成するには、JavaScript アプリケーションソースに以下を追加します。

```
var encodedContextData = AmazonCognitoAdvancedSecurityData.getData(_username,
    _userpoolId, _userPoolClientId);
```

iOS/Swift

コンテキストデータを生成するには、iOS アプリケーションは、[Mobile SDK for iOS](#) モジュール [AWSCognitoIdentityProviderASF](#) を統合できます。

脅威保護のためにエンコードされたコンテキストデータを収集するには、次のスニペットをアプリケーションに追加します。

```
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

コンテキストデータを生成するには、Android アプリケーションは、[Mobile SDK for Android](#) モジュール [aws-android-sdk-cognitoidentityprovider-asf](#) を統合できます。

脅威保護のためにエンコードされたコンテキストデータを収集するには、次のスニペットをアプリケーションに追加します。

```
UserContextDataProvider provider = UserContextDataProvider.getInstance();
// context here is android application context.
```

```
String encodedContextData = provider.getEncodedContextData(context, username,
    userPoolId, userPoolClientId);
```

AWS WAF ウェブ ACL をユーザープールに関連付ける

AWS WAF はウェブアプリケーションファイアウォールです。AWS WAF ウェブアクセスコントロールリスト (ウェブ ACL) を使用すると、クラシックホスト UI、マネージドログイン、Amazon Cognito API サービスエンドポイントへの不要なリクエストからユーザープールを保護できます。ウェブ ACL を使用すると、ユーザープールが応答するすべての HTTPS ウェブリクエストをきめ細かく制御できます。AWS WAF ウェブ ACL の詳細については、「AWS WAF デベロッパーガイド」の「[ウェブアクセスコントロールリスト \(ウェブ ACL\) の管理と使用](#)」を参照してください。

ユーザープールに関連付けられた AWS WAF ウェブ ACL がある場合、Amazon Cognito は選択した非機密ヘッダーとユーザーからのリクエストの内容を に転送します AWS WAF。AWS WAF はリクエストの内容を検査し、ウェブ ACL で指定したルールと比較し、Amazon Cognito にレスポンスを返します。

AWS WAF ウェブ ACLs と Amazon Cognito について知っておくべきこと

- ウェブ ACL ルールは、ユーザー名、パスワード、電話番号、E メールアドレスなど、ユーザープールリクエスト内の個人を特定できる情報 (PII) と一致するように設定することはできません。このデータは利用できません AWS WAF。代わりに、IP アドレス、ブラウザエージェント、リクエストされた API オペレーションなど、ヘッダー、パス、本文のセッションデータと一致するようにウェブ ACL ルールを設定します。
- ウェブ ACL ルールの条件は、ユーザーインタラクティブなマネージドログインページへのユーザーからの最初のリクエストに対してのみ、カスタムブロックレスポンスを返すことができます。後続の接続がカスタムブロックレスポンスの条件に一致すると、カスタムステータスコード、ヘッダー、リダイレクトレスポンスを返しますが、デフォルトのブロックメッセージは返しません。
- によってブロックされたリクエスト AWS WAF は、どのリクエストタイプのリクエストレートクォータにもカウントされません。AWS WAF ハンドラーは、API レベルのスロットリングハンドラーの前に呼び出されます。
- ウェブ ACL を作成すると、ウェブ ACL が完全に伝達されて Amazon Cognito で使用できるようになるまでに少し時間がかかります。伝播時間は数秒から数分までです。は、完全に伝播される前にウェブ ACL を関連付けようと [WAFUnavailableEntityException](#)すると、AWS WAF を返します。
- ユーザープールごとに 1 つのウェブ ACL を関連付けることができます。

- リクエストは、AWS WAF が検査できる範囲を超えたペイロードになる可能性があります が Amazon Cognito [からのオーバーサイズリクエストを処理する方法を設定する方法については、「デベロッパーガイド」の「オーバーサイズリクエストコンポーネントの処理」](#)を参照してください。AWS WAF AWS WAF
- AWS WAF [Fraud Control アカウント乗っ取り防止 \(ATP\)](#) を使用するウェブ ACL を Amazon Cognito ユーザープールに関連付けることはできません。ATP 機能は、AWS-[AWSManagedRulesATPRuleSet](#) マネージドルールグループに含まれています。ウェブ ACL をユーザープールに関連付ける前に、ウェブ ACL がこのマネージドルールグループを使用していないことを確認してください。
- ユーザープールに関連付けられた AWS WAF ウェブ ACL があり、ウェブ ACL のルールが CAPTCHA を提示すると、マネージドログイン TOTP 登録で回復不可能なエラーが発生する可能性があります。CAPTCHA アクションを含み、マネージドログインの TOTP には影響しないルールを作成するには、「[マネージドログイン TOTP MFA 用の AWS WAF ウェブ ACL の設定](#)」を参照してください。

AWS WAF は、次のエンドポイントへのリクエストを検査します。

マネージドログインとクラシックのホストされた UI

[ユーザープールのエンドポイントとマネージドログインのリファレンス](#) 内のすべてのエンドポイントへのリクエスト。

公開 API オペレーション

アプリケーションから Amazon Cognito API へのリクエストのうち、認可に AWS 認証情報を使用しないリクエスト。これには、[InitiateAuth](#)、[RespondToAuthChallenge](#)、[GetUser](#) などの API オペレーションが含まれます。の範囲内にある API オペレーションでは、AWS 認証情報による認証 AWS WAF は必要ありません。これらは、認証されていないか、セッション文字列またはアクセストークンで承認されています。詳細については、「[認可モデル別にグループ化された API オペレーションのリスト](#)」を参照してください。

ウェブ ACL のルールを設定するには、ルールと一致するリクエストに応じて、カウント、許可、ブロック、または CAPTCHA 提示を行うルールアクションを使用できます。詳細については、「AWS WAF デベロッパーガイド」の「[AWS WAF のルール](#)」を参照してください。ルールアクションに応じて、Amazon Cognito からユーザーに返す応答をカスタマイズできます。

⚠ Important

エラー応答をカスタマイズするオプションは、API リクエストを行う方法によって異なります。

- マネージドログインリクエストのエラーコードとレスポンス本文をカスタマイズできます。マネージドログインでは、ユーザーに解決してもらうためだけに CAPTCHA を提示できます。
- Amazon Cognito [ユーザープール API](#) を使用して行うリクエストの場合は、ブロック応答を受信するリクエストの応答本文をカスタマイズできます。カスタムエラーコードを 400 ~ 499 の範囲で指定することもできます。
- AWS Command Line Interface (AWS CLI) と AWS SDKs は、ブロックまたは CAPTCHA レスポンスを生成するリクエストに `ForbiddenException` エラーを返します。

ウェブ ACL をユーザープールに関連付ける

ユーザープールでウェブ ACL を使用するには、AWS Identity and Access Management (IAM) プリンシパルに次の Amazon Cognito および アクセス AWS WAF 許可が必要です。アクセス AWS WAF 許可の詳細については、「AWS WAF デベロッパーガイド」の [AWS WAF 「API アクセス許可」](#) を参照してください。

JSON

```
{
  "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/*"
      ]
    }
  ],
}
```



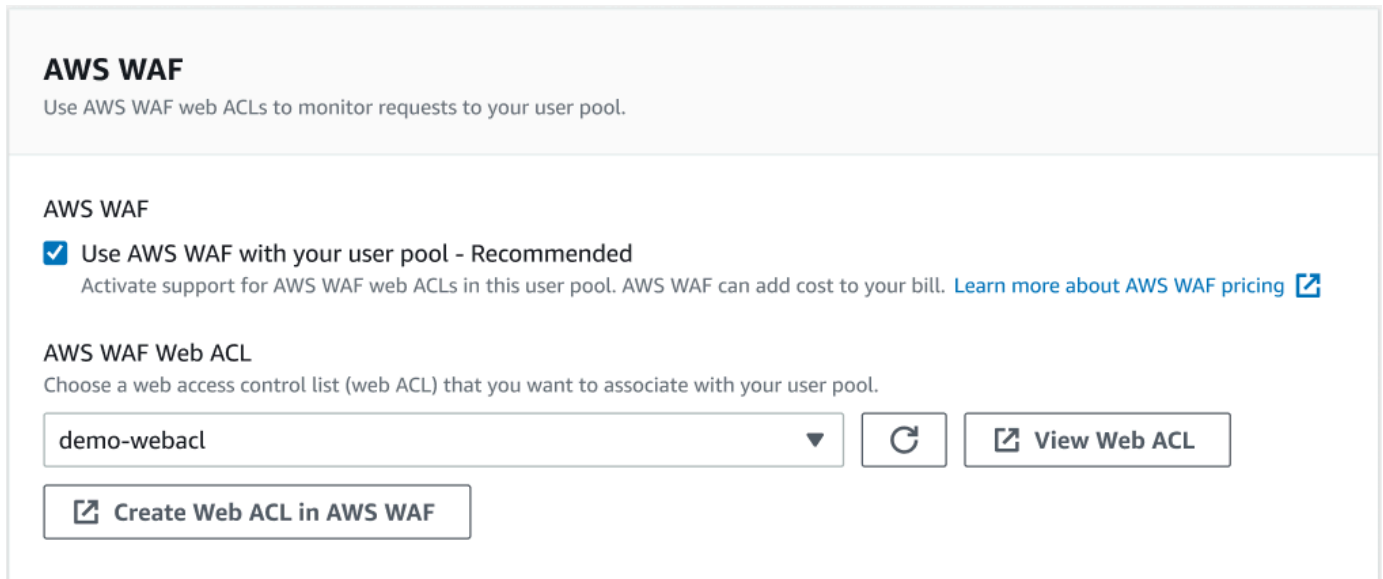
```
{
  "Sid": "AllowWebACLUserPoolWAFv2",
  "Effect": "Allow",
  "Action": [
    "wafv2:ListResourcesForWebACL",
    "wafv2:AssociateWebACL",
    "wafv2:DisassociateWebACL",
    "wafv2:GetWebACLForResource"
  ],
  "Resource": "arn:aws:wafv2:*:123456789012:*/webacl/*/*"
},
{
  "Sid": "DisassociateWebACL1",
  "Effect": "Allow",
  "Action": "wafv2:DisassociateWebACL",
  "Resource": "*"
},
{
  "Sid": "DisassociateWebACL2",
  "Effect": "Allow",
  "Action": [
    "cognito-idp:DisassociateWebACL"
  ],
  "Resource": [
    "arn:aws:cognito-idp:*:123456789012:userpool/*"
  ]
}
]
```

IAM アクセス許可を付与する必要がありますが、リストされているのは、対応する [API オペレーション](#) のない許可のみのアクションです。

ユーザープール AWS WAF に対して をアクティブ化し、ウェブ ACL を関連付けるには

1. [Amazon Cognito コンソール](#) にサインインします。
2. ナビゲーションペインで [User Pools] (ユーザープール) を選択してから、編集するユーザープールを選択します。
3. [セキュリティ] セクションの [AWS WAF] タブを選択します。
4. [編集] を選択します。

5. ユーザープール AWS WAF で を使用する を選択します。



AWS WAF
Use AWS WAF web ACLs to monitor requests to your user pool.

AWS WAF

☒ Use AWS WAF with your user pool - Recommended
Activate support for AWS WAF web ACLs in this user pool. AWS WAF can add cost to your bill. [Learn more about AWS WAF pricing](#)

AWS WAF Web ACL
Choose a web access control list (web ACL) that you want to associate with your user pool.

demo-webacl  [View Web ACL](#)

[Create Web ACL in AWS WAF](#)

6. すでに作成したAWS WAF ウェブ ACL を選択するか、「 でウェブ ACL AWS WAF を作成する」を選択して、 の新しい AWS WAF セッションで作成します AWS マネジメントコンソール。

7. [Save changes] (変更の保存) をクリックします。

ウェブ ACL を AWS Command Line Interface または SDK のユーザープールにプログラムで関連付けるには、AWS WAF API の [AssociateWebACL](#) を使用します。Amazon Cognito には、ウェブ ACL を関連付ける個別の API オペレーションはありません。

AWS WAF ウェブ ACLsテストとログ記録

ウェブ ACL でルールアクションをカウントに設定すると、 はルールに一致するリクエストの数にリクエスト AWS WAF を追加します。ユーザープールでウェブ ACL をテストするには、ルールアクションをカウントに設定し、各ルールに一致するリクエストの量を考慮します。例えば、ルールをブロックアクションに設定した場合に、通常の利用者トラフィックと判断される多数のリクエストと一致するときは、ルールの再設定が必要になる場合があります。詳細については、「AWS WAF デベロッパーガイド」の「[AWS WAF 保護のテストとチューニング](#)」を参照してください。

リクエストヘッダーを Amazon CloudWatch Logs ロググループ、Amazon Simple Storage Service (Amazon S3) バケット、または Amazon Data Firehose に記録する AWS WAF ように を設定することもできます。ユーザープール API で行った Amazon Cognito リクエストは、x-amzn-cognito-client-id と x-amzn-cognito-operation-name で特定できます マネージドログインのリクエ

ストには、x-amzn-cognito-client-id ヘッダーのみが含まれます。詳細については、「AWS WAF デベロッパーガイド」の「[ウェブ ACL トラフィックのログ記録](#)」を参照してください。

AWS WAF ウェブ ACLsは、すべてのユーザープール[機能プラン](#)で使用できます。AWS WAF のセキュリティ機能は、Amazon Cognito の脅威保護を補完します。ユーザープールで両方の機能を有効にすることができます。AWS WAF は、ユーザープールリクエストの検査に対して別途に請求を行います。詳細については、「[AWS WAF 料金](#)」を参照してください。

ログ記録 AWS WAF リクエストデータには、ログをターゲットとするサービスによる追加料金が適用されます。詳細については、「AWS WAF デベロッパーガイド」の「[ウェブ ACL トラフィック情報のログ記録の料金](#)」を参照してください。

ユーザープールの大文字と小文字の区別

で作成した Amazon Cognito ユーザープール AWS マネジメントコンソール では、デフォルトで大文字と小文字は区別されません。ユーザープールが大文字と小文字を区別しない場合、user@example.com と User@example.com は同じユーザーを参照します。ユーザープール内のユーザー名が大文字と小文字が区別しない場合、preferred_username と email 属性でも大文字と小文字が区別されません。

大文字と小文字の区別なしは、属性の入力だけでなく出力にも適用されます。大文字と小文字を区別しないユーザープールにおいて、大文字と小文字が混在する属性値は、ユーザープールのテキスト出力で小文字に変換されます。ユーザープールのテキスト出力の例としては、[userInfo](#) レスポンス、ユーザークエリレスポンス ([GetUser](#) の出力など)、[Lambda トリガー](#)への入力イベントがあります。

ユーザープールの大文字と小文字の区別の設定を考慮するには、代替のユーザー属性に基づいてアプリコードでユーザーを識別します。ユーザー名、優先ユーザー名、または E メールアドレス属性の大文字と小文字はユーザープロフィールによって異なる場合があるため、代わりに sub 属性を参照してください。また、ユーザープールにイミュータブルカスタム属性を作成し、各新規ユーザープロフィールの属性に独自の一意の識別子値を割り当てることもできます。最初にユーザーを作成するときに、作成したイミュータブルカスタム属性に値を書き込むことができます。

Note

ユーザープールの大文字と小文字を区別する設定に関係なく、Amazon Cognito では、SAML または OIDC ID プロバイダー (IdP) のフェデレーテッドユーザーが、大文字と小文字を区別する一意の NameId または sub クレームを渡す必要があります。一意の識別子の小文字と大文字の区別および SAML IdP の詳細については、「[SP が開始した SAML サインインを実装する](#)」を参照してください。

大文字と小文字を区別するユーザープールの作成

AWS Command Line Interface (AWS CLI) および [CreateUserPool](#) などの API オペレーションを使用してリソースを作成する場合は、`BooleanCaseSensitive` パラメータを `true` に設定する必要があります。この設定では、大文字と小文字を区別しないユーザープールが作成されます。値を指定しないと、`CaseSensitive` はデフォルトで `true` に従います。Amazon Cognito コンソールで作成するユーザープールでは、大文字と小文字は区別されません。大文字と小文字を区別するユーザープールを生成するには、`CreateUserPool` オペレーションを使用する必要があります。2020 年 2 月 12 日以前のユーザープールでは、プラットフォームに関係なく、大文字と小文字が区別されていました。

AWS マネジメントコンソールの [サインイン] メニューおよび [DescribeUserPool](#) の `UsernameConfiguration` プロパティで、アカウント内の各ユーザープールの大文字と小文字を区別する設定を確認できます。

新しいユーザープールに移行する

ユーザープロフィール間で競合が発生する可能性があるため、Amazon Cognito ユーザープールで大文字と小文字を区別するから大文字と小文字を区別しないに変更することはできません。代わりに、ユーザーを新しいユーザープールに移行します。ケース関連の競合を解決するには、移行コードをビルドする必要があります。このコードは、競合を検出したときに一意の新しいユーザーを返すか、サインイン試行を拒否する必要があります。大文字と小文字を区別しない新しいユーザープールで、[ユーザー移行の Lambda トリガー](#) を割り当てます。AWS Lambda 関数は、新しい大文字と小文字を区別しないユーザープールにユーザーを作成できます。ユーザーが大文字小文字を区別しないユーザープールでサインインに失敗した場合、Lambda 関数は大文字小文字を区別するユーザープールからユーザーを見つけて、複製します。また、[ForgotPassword](#) イベントで、ユーザーの移行 Lambda トリガーを有効化することもできます。Amazon Cognito は、サインインやパスワード回復アクションからユーザー情報とイベントメタデータを Lambda 関数に渡します。関数が大文字と小文字を区別しないユーザープールに新しいユーザーを作成する際、イベントデータを使用してユーザー名とメールアドレス間で発生する競合を管理することができます。ユーザー名と E メールアドレスが、大文字と小文字を区別するユーザープールでは一意でも、大文字と小文字を区別しないユーザープールでは同一となる場合に競合が発生します。

Amazon Cognito ユーザープール間でユーザーの Lambda 移行トリガーを使用する方法の詳細については、AWS ブログの「[ユーザーの Amazon Cognito ユーザープールへの移行](#)」を参照してください。

ユーザープールの削除保護

管理者が誤ってユーザープールを削除しないようにするには、削除保護を有効にします。削除保護が有効になっている場合は、ユーザープールを削除する前に削除の確認を行う必要があります。でユーザープールを削除すると AWS マネジメントコンソール、削除保護を同時に無効にできます。次の画像に示すように、削除保護を無効にするプロンプトに応じて削除の意思を確認すると、Amazon Cognito がユーザープールを削除します。

Delete user pool ? ×

Before you delete this user pool, first make sure no services or apps rely on it.

 If you delete this user pool, and your app still relies on it, any sign-in and sign-up attempts will fail.

- To delete this user pool, permit Amazon Cognito to also take the following prerequisite actions.
 - ☒ Deactivate deletion protection
- To confirm deletion, enter testUserPool in the field.

testUserPool

Cancel Delete

Amazon Cognito API リクエストを使用してユーザープールを削除する場合は、まず [UpdateUserPool](#) リクエストで DeletionProtection を Inactive に変更する必要があります。削除保護を無効にしないと、Amazon Cognito は InvalidParameterException エラーを返します。削除保護を無効にすると、[DeleteUserPool](#) リクエストでユーザープールを削除できます。

Amazon Cognito は、AWS マネジメントコンソールで新しいユーザープールの作成時に、デフォルトで削除保護を有効にします。CreateUserPool API を使用してユーザープールを作成すると、削除保護はデフォルトで無効になります。AWS CLI または AWS SDK で作成したユーザープールでこの機能を使用するには、DeletionProtectionパラメータを に設定しますTrue。

Amazon Cognito コンソールの [設定] メニューの [削除保護] コンテナで、削除保護ステータスをアクティブ化または非アクティブ化できます。

削除保護を設定するには

1. [Amazon Cognito コンソール](#)に移動します。AWS 認証情報の入力を求められる場合があります。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [設定] メニューを選択し、[削除保護] タブに移動します。[アクティブ化] または [非アクティブ化] を選択します。
5. 次のダイアログで選択を確定します。

ユーザー存在エラー応答の管理

Amazon Cognito は、ユーザープールから返されるエラーレスポンスのカスタマイズをサポートしています。カスタムエラーレスポンスは、ユーザーの作成と認証、パスワードリカバリ、および確認操作に利用できます。

ユーザープールアプリケーションクライアントの `PreventUserExistenceErrors` 設定を使用して、ユーザーの存在に関連するエラーを有効または無効にします。Amazon Cognito ユーザープール API を使用して新しいアプリケーションクライアントを作成する場合、`PreventUserExistenceErrors` はデフォルトで `LEGACY` または無効になっています。Amazon Cognito コンソールでは、オプションのユーザー存在エラーの防止が、`PreventUserExistenceErrors` を `ENABLED` とする設定としてデフォルトで選択されています。`PreventUserExistenceErrors` の設定を更新するには、次のいずれかを実行します。

- [UpdateUserPoolClient](#) API リクエストで `PreventUserExistenceErrors` の値を `ENABLED` と `LEGACY` との間で変更します。
- Amazon Cognito コンソールでアプリケーションクライアントを編集し、ユーザー存在エラーの防止の状態を、選択 (`ENABLED`) と選択解除 (`LEGACY`) との間で変更します。

このプロパティの値が `LEGACY` である場合、ユーザーがユーザープールに存在しないユーザー名でサインインしようとする、アプリケーションクライアントは `UserNotFoundException` エラー応答を返します。

このプロパティの値が `ENABLED` である場合、アプリケーションクライアントは、ユーザープールにユーザーアカウントが存在しないことを `UserNotFoundException` エラーを用いて開示させ

ん。PreventUserExistenceErrors を ENABLED に設定した場合、存在しないユーザー名をリクエストすると、以下のようになります。

- Amazon Cognito は API リクエストに対して非特定の情報で応答します。そうしない場合、その応答から有効なユーザーが存在することが開示される可能性があります。
- Amazon Cognito は、パスワードを忘れた場合のリクエストに対して、さらに USER_SRP_AUTH や CUSTOM_AUTH などの[選択ベースの認証](#) (USER_AUTH) を除く認証フローによる認証リクエストに対して、一般的な認証失敗レスポンスを返します。エラーレスポンスは、ユーザー名またはパスワードが正しくないことを伝えます。
- Amazon Cognito は、選択ベースの認証へのリクエストに対して、ユーザープールで許可されているチャレンジタイプからランダムに選択して応答します。ユーザープールは、パスキー、ワンタイムパスワード、またはパスワードチャレンジを返す場合があります。
- Amazon Cognito のアカウント確認 API とパスワード復旧 API の動作は、シミュレートされた配信メディアにコードが送信されたことを示すレスポンスを返すか、InvalidParameterException エラーを返すかのいずれかです。

次の情報は、PreventUserExistenceErrors が ENABLED に設定されている場合のユーザープールオペレーションの動作を詳しく示しています。

認証とユーザー作成のオペレーション

エラーレスポンスは、ユーザー名パスワード認証とセキュアリモートパスワード (SRP) 認証で設定できます。カスタム認証により、返すエラーをカスタマイズすることもできます。選択ベースの認証は、PreventUserExistenceErrors 設定の影響を受けません。

認証フローでのユーザー存在開示の詳細

選択ベースの認証

USER_AUTH 選択ベースの認証フローの場合、Amazon Cognito は、ユーザープール設定とユーザー属性に応じて、利用可能なプライマリ認証要素からチャレンジを返します。この認証フローでは、パスワード、セキュアリモートパスワード (SRP)、WebAuthn (パスキー)、SMS ワンタイムパスワード (OTP)、または E メール OTP チャレンジを返すことができます。PreventUserExistenceErrors がアクティブな場合、Amazon Cognito は、存在しないユーザーに対して 1 つ以上の利用可能な認証形式を完了するためのチャレンジを発行します。PreventUserExistenceErrors が非アクティブな場合、Amazon Cognito は UserNotFound 例外を返します。

ユーザー名およびパスワード認証

PreventUserExistenceErrors がアクティブな場合、ADMIN_USER_PASSWORD_AUTH 認証フロー、USER_PASSWORD_AUTH 認証フロー、および USER_AUTH の PASSWORD フローは、メッセージ Incorrect username or password とともに NotAuthorizedException を返します。PreventUserExistenceErrors が非アクティブな場合、これらのフローは UserNotFoundException を返します。

セキュアリモートパスワード (SRP) ベースの認証

ベストプラクティスとして、E メールアドレス、電話番号、または優先ユーザー名の [エイリアス属性](#) がないユーザープールでは、PreventUserExistenceErrors を実装するのに USER_SRP_AUTH または USER_AUTH の PASSWORD_SRP フローのみを使用します。エイリアス属性を持つユーザーは、SRP 認証フローでユーザー存在が抑制されない場合があります。ユーザー名パスワード認証フロー (ADMIN_USER_PASSWORD_AUTH、USER_PASSWORD_AUTH、および USER_AUTH PASSWORD チャレンジ) は、エイリアス属性からのユーザー存在を完全に抑制します。

アプリケーションクライアントに知られていないユーザー名を使用して誰かが SRP サインインを試みると、Amazon Cognito は、[RFC 5054](#) で説明されているように、最初のステップでシミュレートされたレスポンスを返します。Amazon Cognito は、同じユーザー名とユーザープールの組み合わせに対して、同じ Salt と [UUID](#) 形式の内部ユーザー ID を返します。パスワードの証明を含む RespondToAuthChallenge API リクエストを送信すると、Amazon Cognito はユーザー名またはパスワードのいずれかが正しくない場合に一般的な NotAuthorizedException エラーを返します。SRP 認証の実装の詳細については、「[永続的なパスワードと安全なペイロードによるサインイン](#)」を参照してください。

Note

検証ベースのエイリアス属性を使用していて、変更不可能なユーザー名の形式が [UUID](#) でない場合は、ユーザー名とパスワードの認証で汎用応答をシミュレートできます。

カスタム認証チャレンジの Lambda トリガー

Amazon Cognito は、CUSTOM_AUTH 認証フローでサインインしようとしたユーザーのユーザー名が見つからない場合、[カスタム認証チャレンジ Lambda トリガー](#) を呼び出します。ユーザーが存在しない場合、入カイベントには UserNotFound という名前のブール型パラメータが true の値とともに含まれます。このパラメータは、カスタム認証アーキテクチャを構成する認証チャ

レンジの作成、定義、検証を行う Lambda 関数にユーザープールから送信するリクエストイベントに表示されます。このインジケータを Lambda 関数のロジックで確認したら、存在しないユーザーのカスタム認証チャレンジをシミュレートできます。

認証前の Lambda トリガー

Amazon Cognito は、サインインしようとしたユーザーのユーザー名が見つからない場合、[認証前トリガー](#)を呼び出します。ユーザーが存在しない場合、入カイベントには `UserNotFound` パラメータが `true` の値とともに含まれます。

ユーザーアカウントの作成に対する `PreventUserExistenceErrors` の影響について以下に説明します。

ユーザー作成フローでのユーザー存在開示の詳細

SignUp

SignUp オペレーションは、ユーザー名がすでに使用されている場合は必ず `UsernameExistsException` を返します。アプリケーションでユーザーをサインアップするときに Amazon Cognito が E メールアドレスと電話番号の `UsernameExistsException` エラーを返さないようにするには、検証ベースのエイリアス属性を使用してください。エイリアスの詳細については、「[ログイン属性のカスタマイズ](#)」を参照してください。

Amazon Cognito が SignUp API リクエストを使用してユーザープール内のユーザーを検出できないようにする方法の例については、「[サインアップ時のメールアドレスと電話番号の UsernameExistsException エラーの防止](#)」を参照してください。

インポート済みユーザー

`PreventUserExistenceErrors` が有効になっている場合は、インポートされたユーザーの認証中、`PasswordResetRequiredException` を返す代わりに、ユーザー名またはパスワードが正しくなかったことを示す `NotAuthorizedException` エラーが返されます。詳細については「[インポートされたユーザーにパスワードをリセットするように要求](#)」を参照してください。

ユーザー移行の Lambda トリガー

Lambda トリガーによって元のイベントコンテキストに空のレスポンスが設定された場合、Amazon Cognito は存在しないユーザーについてシミュレートされたレスポンスを返します。詳細については、「[ユーザー移行の Lambda トリガーを使用したユーザーのインポート](#)」を参照してください。

サインアップ時のメールアドレスと電話番号の `UsernameExistsException` エラーの防止

次の例は、ユーザープールでエイリアス属性を設定するときに、重複する E メールアドレスと電話番号が `SignUp` API リクエストに 응답して `UsernameExistsException` エラーを生成しないようにする方法を示しています。E メールアドレスまたは電話番号をエイリアス属性として使用してユーザープールを作成しておく必要があります。詳細については、「[ユーザープール属性](#)」の「サインイン属性のカスタマイズ」セクションを参照してください。

1. Jie は新しいユーザー名にサインアップし、E メールアドレス `jie@example.com` も提供します。Amazon Cognito がユーザーの E メールアドレスにコードを送信します。

AWS CLI コマンドの例

```
aws cognito-idp sign-up --client-id 1234567890abcdef0 --username jie --password  
PASSWORD --user-attributes Name="email",Value="jie@example.com"
```

レスポンスの例

```
{  
  "UserConfirmed": false,  
  "UserSub": "<subId>",  
  "CodeDeliveryDetails": {  
    "AttributeName": "email",  
    "Destination": "j****@e****",  
    "DeliveryMedium": "EMAIL"  
  }  
}
```

2. Jie は、E メールアドレスの所有権を確認するために送信されたコードを提供します。これで、ユーザーとしての登録は完了です。

AWS CLI コマンドの例

```
aws cognito-idp confirm-sign-up --client-id 1234567890abcdef0 --username=jie --  
confirmation-code xxxxxx
```

3. Shirley は新しいユーザーアカウントを登録し、E メールアドレス `jie@example.com` を提供します。Amazon Cognito は `UsernameExistsException` エラーを返さず、確認コードを Jie の E メールアドレスに送信します。

AWS CLI コマンドの例

```
aws cognito-idp sign-up --client-id 1234567890abcdef0 --username shirley --password  
PASSWORD --user-attributes Name="email",Value="jie@example.com"
```

レスポンスの例

```
{  
  "UserConfirmed": false,  
  "UserSub": "<new subId>",  
  "CodeDeliveryDetails": {  
    "AttributeName": "email",  
    "Destination": "j****@e****",  
    "DeliveryMedium": "EMAIL"  
  }  
}
```

4. 別のシナリオでは、Shirley が jie@example.com の所有権を持っています。Shirley は Amazon Cognito が Jie の E メールアドレスに送信したコードを取得し、アカウントの確認を試みます。

AWS CLI コマンドの例

```
aws cognito-idp confirm-sign-up --client-id 1234567890abcdef0 --username=shirley --  
confirmation-code xxxxxx
```

レスポンスの例

```
An error occurred (AliasExistsException) when calling the ConfirmSignUp operation: An  
account with the email already exists.
```

jie@example.com が既存のユーザーに割り当てられているにもかかわらず、Amazon Cognito は Shirley の `aws cognito-idp sign-up` リクエストにエラーを返しません。Amazon Cognito がエラーレスポンスを返す前に、Shirley は E メールアドレスの所有権を証明する必要があります。エイリアス属性を持つユーザープールでは、この動作により、パブリック SignUp API を使用して、特定の E メールアドレスまたは電話番号を持つユーザーが存在するかどうかを確認できなくなります。

この動作は、次の例に示すように、Amazon Cognito が既存のユーザー名の SignUp リクエストに対して返すレスポンスとは異なります。Shirley はこのレスポンスから、そのユーザー名 jie を持つ

ユーザーが既に存在することを知っていますが、そのユーザーに関連する E メールアドレスや電話番号については知りません。

CLI コマンドの例

```
aws cognito-idp sign-up --client-id lexample23456789 --username jie --password PASSWORD
--user-attributes Name="email",Value="shirley@example.com"
```

レスポンスの例

```
An error occurred (UsernameExistsException) when calling the SignUp operation: User
already exists
```

パスワードのリセットオペレーション

ユーザー存在エラーを防ぐと、Amazon Cognito は、ユーザーパスワードのリセット操作に対して以下の応答を返します。

ForgotPassword

ユーザーが見つからない、非アクティブ化されている、またはパスワードを回復するための検証済みの配信メカニズムがない場合、Amazon Cognito はそのユーザーについてシミュレート済みの配信ミディウムを用いた `CodeDeliveryDetails` を返します。シミュレートされた配信メディアは、入力ユーザー名形式とユーザープールの検証設定によって決まります。

ConfirmForgotPassword

Amazon Cognito は、存在しない、または無効になっているユーザーについて `CodeMismatchException` エラーを返します。ForgotPassword の使用時にコードが要求されない場合、Amazon Cognito は `ExpiredCodeException` エラーを返します。

確認オペレーション

ユーザー存在エラーを防ぐと、Amazon Cognito は、ユーザーの確認および検証の操作に対して以下の応答を返します。

ResendConfirmationCode

Amazon Cognito は、無効化されたユーザー、または存在しないユーザーについて `CodeDeliveryDetails` を返します。Amazon Cognito は、既存ユーザーの E メールまたは電話番号に確認コードを送信します。

ConfirmSignUp

コードの有効期限が切れている場合は、`ExpiredCodeException` が返されます。Amazon Cognito は、ユーザーが承認されていない場合に `NotAuthorizedException` を返します。コードがサーバーが期待するものと一致しない場合、Amazon Cognito は `CodeMismatchException` を返します。

ユーザープールのエンドポイントとマネージドログインのリファレンス

Amazon Cognito には、ユーザープール認証として、ユーザープール API を用いたものと、OAuth 2.0 認可サーバーを用いたものの 2 つのモデルがあります。アプリケーションのバックエンドで AWS SDK を使用して OpenID Connect (OIDC) トークンを取得する場合は、API を使用します。ユーザープールを OIDC プロバイダーとして実装する場合は、認可サーバーを使用します。認可サーバーは、[フェデレーティッドサインイン](#)、[アクセストークンスコープを使用した API および M2M 認可](#)、[マネージドログイン](#)などの機能を追加します。API モデルと OIDC モデルは、ユーザープールレベルまたは[アプリケーションクライアント](#)レベルでの設定により、それぞれ単独で使用することも、併用することもできます。このセクションでは、OIDC モデルの実装に関するリファレンスを示します。これら 2 つの認可モデルの詳細については、「[API、OIDC、マネージドログインページの認証についての理解](#)」を参照してください。

ユーザープールにドメインを割り当てると、Amazon Cognito はここにリストされているパブリックウェブページをアクティブにします。ドメインは、すべてのアプリケーションの中央アクセスポイントとして機能します。これには、ユーザーがサインアップとサインイン ([ログインエンドポイント](#))、およびサインアウト ([ログアウトエンドポイント](#)) を行うことができるマネージドログインが含まれます。これらのリソースの詳細については、「[ユーザープールのマネージドログイン](#)」を参照してください。

これらのページには、ユーザープールがサードパーティーの SAML、OpenID Connect (OIDC)、および OAuth 2.0 ID プロバイダー (IdPs) と通信できるようにするパブリックウェブリソースも含まれています。フェデレーション ID プロバイダーを使用してユーザーをサインインさせる場合、ユーザーは、インタラクティブなマネージドログインの[ログインエンドポイント](#)または OIDC の[認可エンドポイント](#)に対してリクエストを開始する必要があります。認可エンドポイントは、ユーザーをマネージドログインページまたは IdP のサインインページにリダイレクトします。

アプリは、[Amazon Cognito ユーザープール API](#) を使用してローカルユーザーをサインインすることもできます。ローカルユーザーは、外部 IdP を介したフェデレーションなしに、ユーザープールディレクトリにのみ存在します。

Amazon Cognito は、マネージドログインに加えて、Android、iOS、JavaScript などの SDK とも統合されています。SDK は、Amazon Cognito API サービスエンドポイントにユーザープール API オペレーションを実行するためのツールを提供します。サービスエンドポイントの詳細については、「[Amazon Cognito Identity エンドポイントとクォータ](#)」を参照してください。

Warning

Amazon Cognito ドメインのエンドエンティティ証明書または中間 Transport Layer Security (TLS) 証明書を固定しないでください。は、すべてのユーザープールエンドポイントとプレフィックスドメインのすべての証明書 AWS を管理します。Amazon Cognito 証明書をサポートする信頼チェーン内の認証局 (CA) は、動的にローテーションし、更新されます。アプリを中間証明書またはリーフ証明書に固定すると、が証明書を AWS ローテーションするときに、アプリが予告なしに失敗する可能性があります。

代わりに、アプリケーションを利用できるすべての [Amazon ルート証明書](#) に固定します。詳細については、「AWS Certificate Manager ユーザーガイド」で「[証明書のピン留め](#)」のベストプラクティスと推奨事項を参照してください。

トピック

- [ユーザーインタラクティブなマネージドログインとクラシックのホストされた UI のエンドポイント](#)
- [ID プロバイダーと依拠しているパーティーエンドポイント](#)
- [OAuth 2.0 許可](#)
- [認可コード許可での PKCE の使用](#)
- [マネージドログインとフェデレーションのエラーレスポンス](#)

ユーザーインタラクティブなマネージドログインとクラシックのホストされた UI のエンドポイント

ユーザープールにドメインを追加すると、Amazon Cognito は、このセクションのマネージドログインのエンドポイントをアクティブにします。これらは、ユーザーがユーザープールの中核となる認証

操作を完了できるウェブページです。これらには、パスワード管理、多要素認証 (MFA)、および属性検証用のページが含まれます。

マネージドログインを構成するウェブページは、ユーザーとのインタラクティブなユーザーセッション用のフロントエンドウェブアプリケーションです。アプリケーションは、ユーザーのブラウザでマネージドログインを呼び出す必要があります。Amazon Cognito は、この章のウェブページに対するプログラムによるアクセスをサポートしていません。JSON レスポンスを返す [ID プロバイダーと依頼しているパーティーエンドポイント](#) のフェデレーションエンドポイントは、アプリコードで直接クエリできます。[認可エンドポイント](#) は、マネージドログインまたは IdP のサインインページにリダイレクトされます。また、ユーザーのブラウザで開く必要があります。

すべてのユーザープールエンドポイントは、IPv4 および IPv6 のソース IP アドレスからのトラフィックを受け入れます。

このガイドのトピックでは、よく使用されるマネージドログインとクラシックのホストされた UI のエンドポイントについて詳しく説明します。マネージドログインとホストされた UI の違いは外観であり、機能に差はありません。/passkeys/add を除き、すべてのパスは 2 つのバージョンのマネージドログインのブランディング間で共有されます。

Amazon Cognito は、ユーザープールにドメインを割り当てる際に以下のウェブページを利用できるようにします。

マネージドログインのエンドポイント

エンドポイント URL	説明	アクセス方法
<code>https://##### /login</code>	ユーザープールのローカルユーザーとフェデレーションユーザーにサインインします。	認可エンドポイント 、/logout、と /confirmforgotPassword などのエンドポイントからリダイレクトします。「 ログインエンドポイント 」を参照してください。
<code>https://##### /logout</code>	ユーザープールユーザーをサインアウトします。	直接リンク。「 ログアウトエンドポイント 」を参照してください。

エンドポイント URL	説明	アクセス方法
<code>https://##### / ConfirmUser</code>	E メールリンクを選択してユーザーアカウントを検証したユーザーを確認します。	ユーザーが E メールメッセージ内のリンクを選択しました。
<code>https://##### / signup</code>	新規ユーザーをサインアップします。/login ページは、ユーザーが [Sign up] (サインアップ) を選択すると、/signup に移動します。	/oauth2/authorize と同じパラメータを持つ直接リンク。
<code>https://##### / confirm</code>	ユーザープールがサインアップしたユーザーに確認コードを送信すると、ユーザーにコードの入力を求めます。	/signup からのリダイレクトのみ。
<code>https://<i>Your user pool domain</i>/forgotPassword</code>	ユーザーにユーザー名の入力求め、パスワードリセットコードを送信します。/login ページは、ユーザーが [Forgot your password?] (パスワードをお忘れですか?) を選択するとユーザーを /forgotPassword に誘導します。	1. /login のパスワードを忘れた場合のリンクから。 2. /oauth2/authorize と同じパラメータで直接リンクします。
<code>https://<i>Your user pool domain</i>/confirmforgotPassword</code>	ユーザーにパスワードリセットコードと新しいパスワードの入力を求めます。/forgotPassword ページは、ユーザーが [Reset your password] (パスワードのリセット) を選択するとユーザーを /confirmforgotPassword ページに誘導します。	/forgotPassword からのリダイレクトのみ。

エンドポイント URL	説明	アクセス方法
https://##### /resendcode	ユーザープールにサインアップしたユーザーに、新しい確認コードを送信します。	/confirm の [新しいコードを送信] リンクからのリダイレクトのみ。
https://##### /passkeys/add	新しい パスキー を登録します。マネージドログインでのみ使用できます。	- パスキー認証をサポートするアプリケーションクライアントにおける確認後のサインアップフロー内。 /oauth2/authorize と同じパラメータを持つ直接リンク。

トピック

- [マネージドログインのサインインエンドポイント: /login](#)
- [マネージドログインのサインアウトエンドポイント: /logout](#)

マネージドログインのサインインエンドポイント: /login

ログインエンドポイントは認証サーバーであり、[認可エンドポイント](#)からのリダイレクト先です。IDプロバイダーを指定しない場合、マネージドログインへのエントリポイントとなります。ログインエンドポイントへのリダイレクトを生成すると、ログインページにロードされ、ユーザーにクライアントに設定されている認証オプションが提示されます。

Note

ログインエンドポイントは、マネージドログインのコンポーネントです。アプリケーションで、ログインエンドポイントにリダイレクトするフェデレーションページとマネージドログインページを呼び出します。ユーザーがログインエンドポイントに直接アクセスすることはベストプラクティスではありません。

GET/ログイン

/login エンドポイントは、ユーザーの最初のリクエストの HTTPS GET のみをサポートします。アプリは Chrome や Firefox などのブラウザでページを呼び出します。[認可エンドポイント](#) から /login にリダイレクトすると、最初のリクエストで指定したすべてのパラメータが渡されます。ログインエンドポイントは authorize エンドポイントのすべてのリクエストパラメータをサポートします。ログインエンドポイントに直接アクセスすることもできます。ベストプラクティスとしては、すべてのユーザーのセッションを /oauth2/authorize から開始してください。

例 – ユーザーにサインインするよう促す

この例では、ログイン画面が表示されます。

```
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
```

例 – 応答

認証サーバーは認可コードとステートを伴ってアプリにリダイレクトします。サーバーは、コードとステートをフラグメントではなく、クエリ文字列のパラメータで返す必要があります。

```
HTTP/1.1 302 Found
    Location: https://YOUR_APP/redirect_uri?
code=AUTHORIZATION_CODE&state=STATE
```

ユーザーが開始したサインインリクエスト

ユーザーは、/login エンドポイントをロードした後で、ユーザー名とパスワードを入力して [サインイン] を選択できます。これを行うと、GET リクエストと同じヘッダーリクエストパラメータと、ユーザー名、パスワード、およびデバイスのフィンガープリントを含むリクエスト本文で、HTTPS POST リクエストが生成されます。

マネージドログインのサインアウトエンドポイント: /logout

/logout エンドポイントは、リダイレクションエンドポイントです。ユーザーをサインアウトし、アプリケーションクライアントの認可サインアウト URL または /login エンドポイントにリダイレ

クトする必要があります。/logout エンドポイントへの GET リクエストで使用可能なパラメータは、Amazon Cognito マネージドログインのユースケースに合わせて調整されます。

ログアウトエンドポイントは、顧客とのインタラクティブなユーザーセッション用のフロントエンドウェブアプリケーションです。アプリケーションは、このエンドポイントと他のマネージドログインのエンドポイントを、ユーザーのブラウザで呼び出す必要があります。

ユーザーをマネージドログインにリダイレクトして再度サインインさせるには、リクエストに `redirect_uri` パラメータを追加します。`redirect_uri` パラメータを含む `logout` リクエストには、`client_id`、`response_type`、`scope` などの [ログインエンドポイント](#) への後続のリクエスト用のパラメータも含める必要があります。

選択したページにユーザーをリダイレクトするには、アプリケーションに許可されているサインアウト URL を追加します。`logout` エンドポイントへのユーザーのリクエストに、`logout_uri` および `client_id` パラメータを追加します。`logout_uri` の値が、許可されているサインアウト URL の 1 つである場合、Amazon Cognito はユーザーをその URL にリダイレクトします。

SAML 2.0 IdP のシングルログアウト (SLO) を使用すると、Amazon Cognito は、まずユーザーを IdP 設定で定義した SLO エンドポイントにリダイレクトします。IdP がユーザーを `saml2/logout` にリダイレクトすると、Amazon Cognito はリクエストから `redirect_uri` または `logout_uri` へのリダイレクトをもう 1 回返します。詳細については、「[シングルサインアウトで SAML ユーザーをサインアウトする](#)」を参照してください。

ログアウトエンドポイントは、OIDC またはソーシャル ID プロバイダー (IdP) からユーザーにサインアウトしません。外部 IdP を使用してセッションからユーザーをサインアウトするには、そのプロバイダーのサインアウトページにユーザーを誘導します。

GET /logout

/logout エンドポイントは HTTPS GET のみをサポートします。ユーザープールクライアントは通常、このリクエストをシステムブラウザ経由で行います。ブラウザは通常、Android では Custom Chrome Tab、iOS では Safari View Control です。

リクエストパラメータ

`client_id`

アプリのアプリケーション ID。アプリケーション ID を取得するには、ユーザープールにアプリを登録する必要があります。詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

必須。

logout_uri

logout_uri パラメータを使用して、ユーザーをカスタムサインアウトページにリダイレクトします。この値を、ユーザーがサインアウトした後にユーザーをリダイレクトするアプリケーションの[sign-out URL] (サインアウト URL) に設定します。logout_uri は、client_id パラメータでのみ使用してください。詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

logout_uri パラメータを使用して、ユーザーを別のアプリケーションのサインインページにリダイレクトすることもできます。他のアプリケーションのサインインページを、アプリケーションで許可されたコールバック URL として設定します。/logout エンドポイントへのリクエストで、logout_uri パラメータの値を URL エンコードされたサインインページに設定します。

Amazon Cognito では、/logout エンドポイントへのリクエストに logout_uri または redirect_uri パラメータのいずれかが必要です。logout_uri パラメータは、ユーザーを別のウェブサイトへにリダイレクトします。/logout エンドポイントへのリクエストに logout_uri パラメータと redirect_uri パラメータの両方が含まれている場合、Amazon Cognito は logout_uri パラメータのみを使用し、redirect_uri パラメータをオーバーライドします。

nonce

(オプション) リクエストに追加できるランダムな値。指定したノンス値は、Amazon Cognito が発行する ID トークンに含まれます。リプレイ攻撃を防ぐために、アプリは ID トークンの nonce クレームを検査し、生成したものと比較することができます。nonce クレームの詳細については、「[OpenID Connect standard](#)」 (OpenID Connect 標準) の「ID token validation」 (ID トークンの検証) を参照してください。

redirect_uri

redirect_uri パラメータを使用して、ユーザーをサインインページにリダイレクトし、認証を行います。その値を、サインインした後にユーザーをリダイレクトするアプリケーションの[Allowed callback URL] (許可されたコールバック URL) に設定します。/login エンドポイントに渡す client_id、scope、state、response_type パラメータを追加します。

Amazon Cognito では、/logout エンドポイントへのリクエストに logout_uri または redirect_uri パラメータのいずれかが必要です。ユーザーを /login エンドポイントにリダイレクトして再認証し、トークンをアプリケーションに渡すには、redirect_uri パラメータを追加します。/logout エンドポイントへのリクエストに logout_uri パラメータと redirect_uri パラメータの両方が含まれている場合、Amazon Cognito は redirect_uri パラメータをオーバーライドし、logout_uri パラメータのみを処理します。

response_type

ユーザーがサインインした後に Amazon Cognito から受信する OAuth 2.0 レスポンス。code と token は、response_type パラメータの有効な値です。

redirect_uri パラメータを使用する場合は必須です。

state

アプリケーションがリクエストに state パラメータを追加すると、Amazon Cognito は、/oauth2/logout エンドポイントはユーザーをリダイレクトする際に、その値をアプリケーションに返します。

この値をリクエストに追加して [CSRF](#) 攻撃から保護します。

state パラメータの値を、URL でエンコードされた JSON 文字列に設定することはできません。この形式に一致する文字列を state パラメータで渡すには、文字列を base64 にエンコードし、アプリケーション内でデコードします。

redirect_uri パラメータを使用する場合は強くお勧めします。

scope

redirect_uri パラメータを使用してサインアウトした後に Amazon Cognito にリクエストする OAuth 2.0 スcope。Amazon Cognito は、/logout エンドポイントへのリクエストの scope パラメータを使用してユーザーを /login エンドポイントにリダイレクトします。

redirect_uri パラメータを使用する場合は任意。scope パラメータを含めない場合、Amazon Cognito は scope パラメータを使用してユーザーを /login エンドポイントにリダイレクトします。Amazon Cognito がユーザーをリダイレクトし、scope に自動的にデータを入力する場合、パラメータには、アプリケーションに対して許可されているすべての scope が含まれます。

リクエストの例

例: ログアウトしてユーザーをクライアントにリダイレクトする

Amazon Cognito は、リクエストに logout_uri と client_id が含まれている場合、他のすべてのリクエストパラメータを無視して、値が logout_uri である URL にユーザーセッションをリダイレクトします。この URL は、アプリケーションクライアントの承認済みサインアウト URL である必要があります。

次は、サインアウトと <https://www.example.com/welcome> へのリダイレクトのリクエスト例です。

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?
  client_id=1example23456789&
  logout_uri=https%3A%2F%2Fwww.example.com%2Fwelcome
```

例 – ログアウトして別のユーザーとしてサインインするよう指示する

リクエストに `logout_uri` が省略されているが、認可エンドポイントへの正しい形式のリクエストを構成するパラメータが指定されている場合、Amazon Cognito はユーザーをマネージドサインインにリダイレクトします。ログアウトエンドポイントは、元のリクエストのパラメータをリダイレクト先に追加します。

ログアウトリクエストに追加する他のパラメータは、[リクエストパラメータ](#)のリストに含める必要があります。例えば、ログアウトエンドポイントは、`identity_provider` パラメータまたは `idp_identifier` パラメータを使用した自動 IdP リダイレクトをサポートしていません。ログアウトエンドポイントへのリクエストの `redirect_uri` パラメータは、サインアウト URL ではなく、認可エンドポイントにパススルーするサインイン後の URL です。

次に示すのは、ユーザーをサインアウトしてサインインページにリダイレクトし、サインイン後に認可コードを `https://www.example.com` に提供するリクエストの例です。

```
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
```

ID プロバイダーと依拠しているパーティーエンドポイント

フェデレーションエンドポイントは、ユーザープールで使用する認証標準のいずれかの目的を果たすユーザープールエンドポイントです。これには、ID プロバイダーと依拠しているパーティーの両方として、ユーザープールロール用の SAML ACS URLs、OIDC 検出エンドポイント、サービスエンドポイントが含まれます。フェデレーションエンドポイントは、認証フローを開始し、IdP から認証の証拠を受け取り、クライアントにトークンを発行します。これらは IdP、アプリケーション、管理者とインタラクションを行いますが、ユーザーとはインタラクションを行いません。

このページの後に続くページ全体のトピックには、ユーザープールにドメインを追加したときに利用可能になる OAuth 2.0 および OIDC プロバイダーエンドポイントに関する詳細が記載されています。次のチャートは、すべてのフェデレーションエンドポイントのリストです。

[ユーザープールドメイン](#)の例は以下のとおりです。

1. プレフィックスドメイン: `mydomain.auth.us-east-1.amazoncognito.com`
2. カスタムドメイン: `auth.example.com`

ユーザープールのフェデレーションエンドポイント

エンドポイント URL	説明	アクセス方法
<code>https://##### / oauth2/authorize</code>	ユーザーをマネージドログインにリダイレクトするか、IdPでサインインするようにリダイレクトします。	ユーザー認証を開始するためにカスタムブラウザで呼び出されます。「 認可エンドポイント 」を参照してください。
<code>https://##### / oauth2/token</code>	認可コードまたはクライアント認証情報リクエストに基づいてトークンを返します。	トークンを取得するようにアプリケーションからリクエストされました。「 トークンエンドポイント 」を参照してください。
<code>https://##### / oauth2/userInfo</code>	OAuth 2.0 のスコープとアクセストークンのユーザー ID に基づいてユーザー属性を返します。	ユーザープロフィールを取得するようにアプリケーションからリクエストされました。「 userInfo エンドポイント 」を参照してください。
<code>https://##### / oauth2/revoke</code>	更新トークンと関連するアクセストークンを取り消します。	トークンを取り消すようにアプリケーションからリクエストされました。「 エンドポイントの取り消し 」を参照してください。
<code>https://cognito-idp.### #.amazonaws.com/##### ID/.well-known/openid-configu ration</code>	ユーザープールの OIDC アーキテクチャのディレクトリ。 ¹	ユーザープール発行者のメタデータを見つけるようにアプリケーションからリクエストされました。

エンドポイント URL	説明	アクセス方法
<code>https://cognito-idp.#### #.amazonaws.com/##### ID/.well-known/jwks.json</code>	Amazon Cognito トークンの 検証に使用できるパブリック キー。 ²	JWT を検証するようにアプリ ケーションからリクエストさ れました。
<code>https://##### / oauth2/idpresponse</code>	ソーシャル ID プロバイダー は、認可コードを使用して ユーザーをこのエンドポイン トにリダイレクトする必要が あります。Amazon Cognito は、フェデレーションユー ザーを認証する際に、この コードをトークンに引き換え ます。	OIDC IdP サインインから IdP クライアントのコールバック URL としてリダイレクトされ ます。
<code>https://##### /saml2/ idpresponse</code>	SAML 2.0 ID プロバイダーと 統合するためのアサーショ ンコンシューマーサービス (ACS) の URL。	SAML 2.0 IdP から ACS URL として、または IdP が開始し たサインインの発信元ポイン トとしてリダイレクトされま す ³ 。
<code>https://##### / saml2/logout</code>	SAML 2.0 ID プロバイダーと 統合するための シングルログ アウト (SLO) の URL。	シングルログアウト (SLO) URL として SAML 2.0 IdP からリダイレクトされまし た。POST バインディングの みを受け入れます。

¹ openid-configurationドキュメントは、エンドポイントを OIDC および OAuth2 仕様に準拠させるための追加情報を反映するために、随時更新される場合があります。

² jwks.json JSON ファイルは、新しいパブリックトークン署名キーを反映するために、随時更新される場合があります。

³ IdP が開始した SAML サインインの詳細については、「」を参照してください[IdP が開始した SAML サインインを実装する](#)。

OpenID および OAuth 標準の詳細については、「[OpenID 1.0](#)」および「[OAuth 2.0](#)」を参照してください。

トピック

- [リダイレクトエンドポイントと認可エンドポイント](#)
- [トークン発行者エンドポイント](#)
- [ユーザー属性エンドポイント](#)
- [トークン取り消しエンドポイント](#)
- [IdP SAML アサーションエンドポイント](#)

リダイレクトエンドポイントと認可エンドポイント

/oauth2/authorize エンドポイントは、2 つのリダイレクト先をサポートするリダイレクトエンドポイントです。URL に identity_provider または idp_identifier のパラメータを含めると、ユーザーはその ID プロバイダー (IdP) のサインインページにそのままリダイレクトされます。それ以外の場合は、リクエストに含まれるものと同じ URL パラメータを持つ [ログインエンドポイント](#) にリダイレクトされます。

認可エンドポイントは、マネージドログインまたは IdP サインインページのどちらかにリダイレクトされます。このエンドポイントにおけるユーザーセッションの送信先は、ユーザーがブラウザで直接操作する必要があるウェブページです。

認可エンドポイントを使用するには、ユーザープールに以下のユーザープールの詳細に関する情報を提供するパラメータを使用して、ユーザーのブラウザを /oauth2/authorize で呼び出します。

- サインインするアプリクライアント。
- 終了させたいコールバック URL。
- ユーザーのアクセストークンでリクエストする OAuth 2.0 スコープ。
- サインインに使用するサードパーティーの IdP (任意)。

また、Amazon Cognito が受信クレームの検証に使用する state および nonce パラメータを提供することもできます。

GET /oauth2/authorize

/oauth2/authorize エンドポイントは HTTPS GET のみをサポートします。アプリは通常、このリクエストをユーザーのブラウザで開始します。/oauth2/authorize エンドポイントへのリクエストは、HTTPS でのみ行うことができます。

OpenID Connect (OIDC) 標準における認可エンドポイントの定義の詳細については、「[Authorization Endpoint](#)」(認可エンドポイント) を参照してください。

リクエストパラメータ

response_type

必須。

レスポンスのタイプ。code または token を指定する必要があります。

response_type が code のリクエストに成功すると、認可コード付与を返します。認可コード付与とは、Amazon Cognito がリダイレクト URL に追加する code パラメータです。アプリは [トークンエンドポイント](#) と、アクセス、ID、更新の各トークンとコードを交換することができます。セキュリティ上のベストプラクティスとして、またユーザーに更新トークンを受け取るには、アプリで認可コード付与を使用します。

response_type が token のリクエストに成功すると、暗黙的な付与を返します。暗黙的な付与とは、Amazon Cognito がリダイレクト URL に追加する ID とアクセストークンのことです。暗黙的な付与は、トークンと潜在的な識別情報をユーザーに公開するため、安全性が低くなります。アプリクライアントの設定で、暗黙的な付与のサポートを無効にできます。

client_id

必須。

アプリクライアント ID。

client_id の値は、リクエストを行うユーザープール内のアプリクライアントの ID である必要があります。アプリクライアントは、Amazon Cognito ローカルユーザーまたは少なくとも 1 つのサードパーティー IdP によるサインインをサポートしている必要があります。

redirect_uri

必須。

Amazon Cognito がユーザーを認証した後、認証サーバーによってブラウザがリダイレクトされる URL です。

リダイレクト Uniform Resource Identifier (URI) には次の属性が必要です。

- 絶対 URI である。
- URI を事前にクライアントに登録しておく必要があります。
- フラグメントコンポーネントが含まれていない。

「[OAuth 2.0 - リダイレクトエンドポイント](#)」を参照してください。

Amazon Cognito では、テスト目的のコールバック URL として設定できる `http://localhost` を除き、リダイレクト URI に HTTPS を使用することが必要です。

Amazon Cognito では、`myapp://example` のようなアプリのコールバック URL もサポートしています。

state

オプション、推奨。

アプリがリクエストに `state` パラメータを追加すると、Amazon Cognito は `/oauth2/authorize` エンドポイントはユーザーをリダイレクトする際に、その値をアプリに返します。

この値をリクエストに追加して [CSRF](#) 攻撃から保護します。

`state` パラメータの値を、URL でエンコードされた JSON 文字列に設定することはできません。この形式に一致する文字列を `state` パラメータで渡すには、文字列を base64 にエンコードし、アプリケーション内でデコードします。

identity_provider

オプション。

このパラメータを追加して、マネージドログインをバイパスし、ユーザーをプロバイダーのサインインページにリダイレクトします。`identity_provider` パラメータの値はユーザープールに表示される ID プロバイダー (IdP) の名前です。

- ソーシャルプロバイダーの場合、`identity_provider` 値として Facebook、Google、LoginWithAmazon、SignInWithApple を使用できます。
- Amazon Cognito ユーザープールの場合、値 `COGNITO` を使用します。
- SAML 2.0 および OpenID Connect (OIDC) ID プロバイダー (IdP) の場合は、ユーザープールで IdP に割り当てた名前を使用します。

idp_identifier

オプション。

このパラメータを追加して、`identity_provider` 名の代替名を持つプロバイダーにリダイレクトします。Amazon Cognito コンソールの [ソーシャルプロバイダーと外部プロバイダー] メニューから、SAML 2.0 と OIDC IdP の識別子を入力できます。

scope

オプション。

クライアントに関連付けられているシステム予約されたスコープ、またはカスタムスコープの組み合わせにすることができます。スコープはスペースで区切る必要があります。システム予約されたスコープは `openid`、`email`、`phone`、`profile`、および `aws.cognito.signin.user.admin` です。使用されるスコープは、いずれもクライアントに関連付けられている必要があります、関連付けられていなければ、ランタイム時に無視されます。

クライアントがスコープをリクエストしない場合、認証サーバーはクライアントに関連付けられているすべてのスコープを使用します。

ID トークンは `openid` スコープがリクエストされた場合にのみ返されます。Amazon Cognito ユーザープールに対してアクセストークンを使用できるのは、`aws.cognito.signin.user.admin` スコープがリクエストされている場合のみです。`phone`、`email`、および `profile` スコープは、`openid` スコープがリクエストされた場合にのみリクエストできます。これらのスコープは ID トークン内でクレームを記述します。

code_challenge_method

オプション。

チャレンジの生成に使用したハッシュプロトコル。[PKCE RFC](#) では S256 および plain の 2 つのメソッドが定義されていますが、Amazon Cognito 認証サーバーでは S256 のみがサポートされています。

code_challenge

オプション。

`code_verifier` から生成した Proof of Key Code Exchange (PKCE) チャレンジ。詳細については、「[認可コード許可での PKCE の使用](#)」を参照してください。

`code_challenge_method` パラメータを指定した場合にのみ必須です。

nonce

オプション。

リクエストに追加できるランダムな値。指定したノンス値は、Amazon Cognito が発行する ID トークンに含まれます。リプレイ攻撃を防ぐために、アプリは ID トークンの nonce クレームを検査し、生成したものと比較することができます。nonce クレームの詳細については、「[OpenID Connect standard](#)」(OpenID Connect 標準) の「ID token validation」(ID トークンの検証) を参照してください。

lang

オプション。

ユーザーインタラクティブページを表示する言語。マネージドログインページはローカライズできますが、ホストされた UI (クラシック) ページはローカライズできません。詳細については、「[マネージドログインのローカリゼーション](#)」を参照してください。

login_hint

オプション。

認可サーバーに渡すユーザー名プロンプト。ユーザーからユーザー名、E メールアドレス、または電話番号を収集し、送信先プロバイダーがユーザーのサインイン名を事前に入力できるようにできます。oauth2/authorize エンドポイントに login_hint パラメータを送信し、idp_identifier または identity_provider パラメータを送信しない場合、マネージドログインはユーザー名フィールドにヒント値を入力します。また、このパラメータを [ログインエンドポイント](#) に渡し、ユーザー名値を自動的に入力することもできます。

認可リクエストが OIDC IdP または Google へのリダイレクトを呼び出すと、Amazon Cognito はそのサードパーティーオソライザーへのリクエストに、login_hint パラメータを追加します。ログインヒントを SAML、Apple、Login With Amazon、または Facebook (Meta) の IdP に転送することはできません。

prompt

オプション。

既存のセッションの認証動作を制御する OIDC パラメータ。クラシックのホストされた UI ではなく、マネージドログインのブランディングバージョンでのみ使用できます。OIDC 仕様の詳細については、「[認証リクエスト](#)」を参照してください。値 none と login は、ユーザープールの認証動作に影響します。

ユーザーがサードパーティプロバイダーによる認証を選択した場合、Amazon Cognito は、prompt のすべての値 (none を除く) を IdP に転送します。これは、ユーザーがアクセスする URL に identity_provider パラメータまたは idp_identifier パラメータが含まれてい

る場合、または認可サーバーがユーザーを[ログインエンドポイント](#)にリダイレクトし、ユーザーが使用可能なボタンから IdP を選択した場合に当てはまります。

プロンプトパラメータ値

prompt=none

Amazon Cognito は、有効な認証済みセッションを持つユーザーの認証をサイレントに続けます。このプロンプトを使用すると、ユーザーはユーザープール内の異なるアプリケーションクライアント間でサイレント認証を行うことができます。ユーザーがまだ認証されていない場合、認可サーバーは `login_required` エラーを返します。

prompt=login

Amazon Cognito では、既存のセッションがある場合でも、ユーザーに再認証を求めます。ユーザーの ID を再検証する場合は、この値を送信します。既存のセッションを持つ認証済みユーザーは、そのセッションを無効にすることなく、サインインに戻ることができます。既存のセッションを持つユーザーが再度サインインすると、Amazon Cognito は新しいセッション Cookie をユーザーに割り当てます。このパラメータは IdP に転送することもできます。このパラメータを受け入れる IdP は、ユーザーに新しい認証試行もリクエストします。

prompt=select_account

この値はローカルサインインには影響せず、IdP にリダイレクトするリクエストで送信する必要があります。認可リクエストに含めると、このパラメータは IdP リダイレクト先の URL パスに `prompt=select_account` を追加します。このパラメータを IdP がサポートしている場合、ユーザーはログインに使用するアカウントを選択するよう求められます。

prompt=consent

この値はローカルサインインには影響せず、IdP にリダイレクトするリクエストで送信する必要があります。認可リクエストに含めると、このパラメータは IdP リダイレクト先の URL パスに `prompt=consent` を追加します。このパラメータを IdP がサポートしている場合、IdP は、ユーザープールへのリダイレクト前に、ユーザーの同意をリクエストします。

リクエストから `prompt` パラメータを省略すると、マネージドログインはデフォルトの動作に従います。ブラウザに有効なマネージドログインセッション Cookie がない限り、ユーザーはサインインする必要があります。`prompt` の複数の値をスペース文字で区切って組み合わせることができます (例: `prompt=login consent`)。

resource

オプション。

aud クレーム内のアクセストークンにバインドするリソースの識別子。このパラメータを含めると、Amazon Cognito は値が URL であることを確認し、結果のアクセストークンのオーディエンスを、リクエストされたリソースに設定します。ユーザープールの [リソースサーバー](#) は、URL 形式の識別子、または任意の URL を使用してリクエストできます。このパラメータの値は、https://、http://localhost、またはカスタム URL スキーム (myapp:// など) で始める必要があります。

リソースバインディングは [RFC 8707](#) で定義されています。リソースサーバーとリソースバインディングの詳細については、「[リソースバインディング](#)」を参照してください。

例: 認証コード付与

これは、認可コード付与のリクエストの例です。

次のリクエストは、ユーザーが redirect_uri 送信先でアプリケーションに渡す認可コードを取得するセッションを開始します。このセッションでは、ユーザー属性用スコープと、Amazon Cognito セルフサービス API オペレーションへのアクセス用スコープをリクエストします。

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=openid+profile+aws.cognito.signin.user.admin
```

Amazon Cognito 認証サーバーは、認可コードとステートを伴ってリダイレクトしアプリに戻ります。認可コードは 5 分間有効です。

```
HTTP/1.1 302 Found
Location: https://www.example.com?code=a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111&state=abcdefg
```

例: PKCE を使用した認証コード付与

このフロー例では、[PKCE](#) を使用して認証コード付与を実行します。

このリクエストでは、code_challenge パラメータを追加します。トークンのコード交換を完了するには、/oauth2/token エンドポイントへのリクエストに code_verifier パラメータを含める必要があります。

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=aws.cognito.signin.user.admin&
code_challenge_method=S256&
code_challenge=a1b2c3d4...
```

認可サーバーは、認証コードと状態をアプリケーションにリダイレクトします。アプリケーションは認証コードを処理し、トークンと交換します。

```
HTTP/1.1 302 Found
Location: https://www.example.com?code=a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111&state=abcdefg
```

例: **prompt=login** で再認証を要求する

次のリクエストでは、既存のセッションがある場合でも、ユーザーに再認証を求める **prompt=login** パラメータを追加します。

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=openid+profile+aws.cognito.signin.user.admin&
prompt=login
```

認可サーバーは、[ログインエンドポイント](#)にリダイレクトし、再認証を要求します。

```
HTTP/1.1 302 Found Location: https://mydomain.auth.us-east-1.amazoncognito.com/
login?response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com&state=abcdefg&scope=openid+profile
+aws.cognito.signin.user.admin&prompt=login
```

例: **prompt=none** によるサイレント認証

次のリクエストでは、ユーザーが有効なセッションを持っているかどうかをサイレントに確認する **prompt=none** パラメータを追加します。


```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=openid+profile+aws.cognito.signin.user.admin&
prompt=none
```

有効なセッションが存在しない場合、認可サーバーはリダイレクト URI にエラーを返します。

```
HTTP/1.1 302 Found Location: https://www.example.com?error=login_required&state=abcdefg
```

有効なセッションが存在する場合、認可サーバーは認証コードを返します。

```
HTTP/1.1 302 Found Location: https://www.example.com?
code=AUTHORIZATION_CODE&state=abcdefg
```

例: リソースバインディングによる認証コード付与

次のリクエストは、アクセストークンを特定のリソースサーバーにバインドする `resource` パラメータを追加します。結果としてのアクセストークンは、それが認証済みユーザーのリクエストのオーディエンスであることを、ターゲット API で検証するための条件を作成します。

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=solar-system-data-api.example.com/asteroids.add&
resource=https://solar-system-data-api.example.com
```

認可サーバーは、`aud` クレームとして `https://solar-system-data-api.example.com` を持つアクセストークンを生成する認可コードを返します。

```
HTTP/1.1 302 Found Location: https://www.example.com?
code=AUTHORIZATION_CODE&state=abcdefg
```

例: `openid` スコープなしのトークンの (暗黙的) 付与

このフロー例では、暗黙的な付与を生成し、JWT をユーザーのセッションに直接返します。

次のリクエストでは、認可サーバーに対して暗黙的な付与を求めます。ユーザープロファイルのセルフサービスオペレーションを認可するアクセストークン内のスコープをリクエストします。

```
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
```

認可サーバは、アクセストークンのみをアプリケーションにリダイレクトします。openid スコープがリクエストされなかったため、Amazon Cognito は ID トークンを返しません。また、Amazon Cognito はこのフローで更新トークンを返しません。

```
HTTP/1.1 302 Found
Location: https://example.com/
callback#access_token=eyJra456defEXAMPLE&token_type=bearer&expires_in=3600&state=STATE
```

例: **openid** スコープ付きトークンの (暗黙的) 付与

このフロー例では、暗黙的な付与を生成し、トークンをユーザーのブラウザに返します。

次のリクエストでは、認可サーバーに対して暗黙的な付与を求めます。ユーザー属性とセルフサービスオペレーションへのアクセスを認可するアクセストークン内のスコープをリクエストします。

```
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
```

認可サーバーは、アクセストークンと ID トークンをアプリケーションにリダイレクトします (openid スコープが含まれていたため)。

```
HTTP/1.1 302 Found
Location: https://
www.example.com#id_token=eyJra67890EXAMPLE&access_token=eyJra12345EXAMPLE&token_type=bearer&exp
```

否定応答の例

Amazon Cognito はリクエストを拒否する場合があります。否定リクエストには、HTTP エラーコードと、リクエストパラメータの修正に使用できる説明が付属しています。否定応答の例を次に示します。

- `client_id` と `redirect_uri` が有効であるが、リクエストパラメータが正しくフォーマットされていない場合、認証サーバーはエラーをクライアントの `redirect_uri` にリダイレクトし、URL パラメータにエラーメッセージを追加します。形式が正しくない場合の例を以下に示します。
- リクエストには `response_type` パラメータが含まれていません。
- 認可リクエストは `code_challenge` パラメータを提供しましたが、`code_challenge_method` パラメータは提供していません。
- `code_challenge_method` パラメータの値が S256 ではありません。

以下は、形式が正しくないリクエスト例に対する応答です。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request
```

- クライアントが `response_type` で `code` または `token` をリクエストしたが、これらのリクエストのアクセス許可を持っていないという場合は、以下にあるように、Amazon Cognito 認可サーバーが `unauthorized_client` をクライアントの `redirect_uri` に返します。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=unauthorized_client
```

- クライアントが無効、不明、または有効ではない形式のスコープをリクエストする場合は、以下にあるように、Amazon Cognito 認可サーバーが `invalid_scope` をクライアントの `redirect_uri` に返します。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_scope
```

- サーバーで予期しないエラーが発生した場合は、認証サーバーがクライアントの `redirect_uri` に `server_error` を返します。HTTP 500 エラーはクライアントに送信されないため、ユーザーのブラウザにエラーが表示されません。認可サーバーは次のエラーを返します。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=server_error
```

- Amazon Cognito がサードパーティー ID プロバイダーとのフェデレーションを通じて認証するとき、以下のような接続問題が発生する場合があります。

- IdP からトークンをリクエストしているときに接続タイムアウトが発生した場合、認証サーバーは以下のようにエラーをクライアントの `redirect_uri` にリダイレクトします。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=Timeout+occurred+in+calling+IdP+token
+endpoint
```

- ID トークン検証のための `jwks_uri` エンドポイントの呼び出し時に接続タイムアウトが発生する場合、認証サーバーは以下のようにエラーをクライアントの `redirect_uri` にリダイレクトします。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=error_description=Timeout+in+calling+jwks
+uri
```

- サードパーティー IdP による認証がなされた場合、プロバイダーはエラー応答を返すことがあります。これは、設定エラーのほか、次のような理由が原因である可能性があります。
- 他のプロバイダーからエラーレスポンスを受け取った場合、認証サーバーは以下のようにエラーをクライアントの `redirect_uri` にリダイレクトします。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=[IdP name]+Error+--[status code]+error
getting token
```

- Google からエラーレスポンスを受け取った場合、認証サーバーは以下のようにエラーをクライアントの `redirect_uri` にリダイレクトします。

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=Google+Error+--[status code]+[Google-
provided error code]
```

- Amazon Cognito が外部 IdP への接続中に通信プロトコルで例外を検出した場合、認証サーバーは次のいずれかのメッセージでエラーをクライアントの `redirect_uri` にリダイレクトします。

```
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
```

トークン発行者エンドポイント

/oauth2/token の OAuth 2.0 [トークンエンドポイント](#) は、認証コードとクライアント認証情報の付与フローを完了するアプリケーションに JSON ウェブトークン (JWT) を発行します。これらのトークンは、ユーザープールによる認証の最終結果です。これには、ユーザー (ID トークン)、ユーザーのアクセスレベル (アクセストークン)、サインインセッションを継続するユーザーの権限 (更新トークン) に関する情報が含まれます。OpenID Connect (OIDC) 依拠しているパーティライブラリは、このエンドポイントへのリクエストとこのエンドポイントからの応答ペイロードを処理します。トークンは、検証可能な認証証拠、プロフィール情報、バックエンドシステムへのアクセスメカニズムを提供します。

ユーザープール OAuth 2.0 認可サーバーは、トークンエンドポイントから以下のタイプのセッションに JSON ウェブトークン (JWT) を発行します。

1. 認可コード付与のリクエストを完了したユーザー。コードの引き換えに成功すると、ID、アクセス、更新の各トークンが返されます。
2. クライアント認証情報の付与を完了したマシンツーマシン (M2M) セッション。クライアントシークレットによる認可が成功すると、アクセストークンが返されます。
3. 以前にサインインして、更新トークンを受け取っていたユーザー。更新トークン認証は、新しい ID トークンとアクセストークンを返します。

Note

マネージドログインまたはフェデレーションを通じて認証コード付与を使用してサインインするユーザーは、トークンエンドポイントからトークンを常に更新できます。API オペレーションの `InitiateAuth` と `AdminInitiateAuth` でサインインし、[記憶されているデバイス](#) がユーザープールでアクティブでないときにトークンエンドポイントでトークンを更新できるユーザー。記憶されたデバイスがアクティブな場合は、アプリケーションクライアントで[関連する API または SDK トークン更新オペレーション](#)を使用してトークンを更新します。

ユーザープールにドメインを追加すると、トークンエンドポイントが公開されます。HTTP POST リクエストを受け入れます。アプリケーションのセキュリティを最適化するには、認可コードのサインインイベントで PKCE を使用します。PKCE は、認可コードを渡すユーザーが、認証したユーザーと同じであることを確認します。PKCE の詳細については、[IETF RFC 7636](#) を参照してください。

ユーザープールのアプリケーションクライアントと付与タイプ、クライアントシークレット、許可されているスコープ、クライアント ID の詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。M2M 認可、クライアント認証情報の付与、アクセストークンスコープによる認可の詳細については、「[スコープ、M2M、リソースサーバー](#)」を参照してください。

アクセストークンからユーザーに関する情報を取得するには、それを [userInfo エンドポイント](#) または [GetUser](#) API リクエストに渡します。アクセストークンには、これらのリクエストに適したスコープが含まれている必要があります。

トークンエンドポイントへの POST リクエストをフォーマットする

/oauth2/token エンドポイントは HTTPS POST のみをサポートします。このエンドポイントはユーザーインタラクティブではありません。トークンリクエストは、アプリケーションの [OpenID Connect \(OIDC\) ライブラリ](#) を使用して処理されます。

トークンエンドポイントは client_secret_basic と client_secret_post をサポートしています。OIDC 仕様の詳細については、「[クライアント認証](#)」を参照してください。OpenID Connect 仕様のトークンエンドポイントの詳細については、「[トークンエンドポイント](#)」を参照してください。

ヘッダーのリクエストパラメータ

リクエストのヘッダーで、以下のパラメータをトークンエンドポイントに渡すことができます。

Authorization

クライアントがシークレットで発行された場合、クライアントは、認可ヘッダー内の client_id および client_secret を client_secret_basic HTTP 認可として渡す必要があります。また、client_id と client_secret を client_secret_post 認可としてリクエスト本文に含めることもできます。

認可ヘッダー文字列は [Basic](#) Base64Encode(client_id:client_secret) です。次の例は、アプリケーションクライアント djc98u3jiedmi283eu928 のクライアントシークレット abcdef01234567890 が付いた認可ヘッダーで、文字列 djc98u3jiedmi283eu928:abcdef01234567890 の Base64 エンコードされたバージョンを使用するものです。

```
Authorization: Basic ZGpjOThtM2ppZWRTaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw
```

Content-Type

このパラメータの値を 'application/x-www-form-urlencoded' に設定します。

本文のリクエストパラメータ

トークンエンドポイントへのリクエスト本文で x-www-form-urlencoded 形式を使用してリクエストできるパラメータは、以下のとおりです。

grant_type

必須。

リクエストする OIDC 付与のタイプ。

authorization_code、refresh_token、または client_credentials を指定する必要があります。以下の条件下で、トークンエンドポイントに対してカスタムスコープのアクセストークンをリクエストできます。

- アプリケーションクライアント設定で、リクエストされたスコープを有効にしました。
- クライアントシークレットを使用してアプリケーションクライアントを設定しました。
- アプリケーションクライアントでクライアント認証情報の付与を有効にします。

Note

トークンエンドポイントは、grant_type が authorization_code である場合にのみ、更新トークンを返します。

client_id

オプション。Authorization ヘッダーでアプリケーションクライアント ID を指定する場合は、必須ではありません。

ユーザープール内のアプリクライアントの ID。ユーザーを認証したものと同一アプリケーションクライアントを指定します。

クライアントがパブリックで、シークレットがない場合、または client_secret_post 認可で client_secret を使用している場合は、このパラメータを指定する必要があります。

client_secret

オプション。*Authorization* ヘッダーでクライアントシークレットを指定する場合や、アプリケーションクライアントにシークレットがない場合は、必須ではありません。

アプリケーションクライアントにシークレットがある場合は、`client_secret_post` 認可用のアプリケーションクライアントシークレット。

scope

オプション。

アプリケーションクライアントに関連付けられている任意のスキープの組み合わせを指定できます。Amazon Cognito は、リクエストされたアプリケーションクライアントで許可されていないリクエスト内のスキープを無視します。このリクエストパラメータを指定しない場合、認可サーバーは、アプリケーションクライアント設定で有効にしたすべての認可スキープを含むアクセストークン `scope` クレームを返します。リクエストされたアプリケーションクライアントで許可されているスキープとして、標準スキープ、リソースサーバーからのカスタムスキープ、`aws.cognito.signin.user.admin` ユーザーセルフサービススキープのいずれかをリクエストできます。

redirect_uri

オプション。クライアント認証情報の付与には必須ではありません。

`/oauth2/authorize` で `authorization_code` を取得するために使用されたものと同じ `redirect_uri` である必要があります。

`grant_type` が `authorization_code` である場合、このパラメータを指定する必要があります。

refresh_token

オプション。ユーザーが既に更新トークンを持っていて、新しい ID トークンとアクセストークンを取得する場合にのみ使用します。

ユーザーのセッション用に新しいアクセストークンと ID トークンを生成するには、`refresh_token` の値を、リクエストされたアプリケーションクライアントが発行した有効な更新トークンに設定します。

[更新トークンのローテーション](#) がアクティブな場合は、新しい ID トークンおよびアクセストークンとともに新しい更新トークンを返します。それ以外の場合は、ID トークンとアクセストーク

ンのみを返します。元のアクセストークンが [API リソースにバインド](#)されていた場合、新しいアクセストークンは、リクエストされた API URL を aud クレーム内に保持します。

code

オプション。認証コード付与でのみ必須です。

認証コード付与からの認証コード。認可リクエストに authorization_code の grant_type が含まれている場合は、このパラメータを指定する必要があります。

aws_client_metadata

オプション。

[Machine to Machine \(M2M\)](#) 認可フローで、[トークン生成前の Lambda トリガー](#)に渡す情報。アプリケーションはセッションに関するコンテキスト情報を収集し、このパラメータで渡すことができます。URL エンコードされた JSON 形式で aws_client_metadata を渡すと、Amazon Cognito はそれをトリガー Lambda 関数への入力イベントに含めます。トークン生成前トリガーイベントバージョンまたはグローバル Lambda トリガーバージョンは、バージョン 3 以降に設定する必要があります。Amazon Cognito は認証コードとクライアント認証情報の M2M フローでこのエンドポイントへのリクエストを受け入れますが、ユーザープールは、クライアント認証情報リクエストのトークン生成前トリガーにのみ aws_client_metadata を渡します。

code_verifier

オプション。最初の認可リクエストで code_challenge_method パラメータと code_challenge パラメータを指定した場合のみ必須です。

[PKCE](#) を使用した認証コード付与リクエストで、アプリケーションが計算した code_challenge の発行元である生成済みコード検証子。

トークン用の認可コードの交換

次のリクエストは、認証コード付与による認証後に ID、アクセストークン、更新トークンを正常に生成します。リクエストは、クライアントシークレットを client_secret_basic の形式で Authorization ヘッダーに渡します。

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token&
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpjOTh1M2ppZWRTaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=authorization_code&
```

```
client_id=1example23456789&
code=AUTHORIZATION_CODE&
redirect_uri=com.myclientapp://myclient/redirect
```

レスポンスは、追加のメタデータとともに、新しい ID、アクセストークン、更新トークンをユーザーに発行します。

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "eyJra1example",
  "id_token": "eyJra2example",
  "refresh_token": "eyJj3example",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

基本認可によるクライアント認証情報

M2M アプリケーションからの次のリクエストは、クライアント認証情報の付与をリクエストします。クライアント認証情報にはクライアントシークレットが必要であるため、リクエストはアプリケーションクライアント ID とシークレットから派生した Authorization ヘッダーを使用して認可されます。リクエストにより、リクエストされた 2 つのスコープを持つアクセストークンが生成されます。リクエストには、IP アドレス情報を提供するクライアントメタデータと、付与対象であるユーザーに発行されたトークンも含まれます。Amazon Cognito は、クライアントメタデータをトークン生成前の Lambda トリガーに渡します。

```
POST https://mydomain.auth.us-east-1.amazonaws.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWRTaTI4M2V10TI40mFiY2RLZjAxMjM0NTY3ODkw

grant_type=client_credentials&
client_id=1example23456789&
scope=resourceServerIdentifier1%2Fscope1%20resourceServerIdentifier2%2Fscope2&
&aws_client_metadata=%7B%22onBehalfOfToken%22%3A%22eyJra789ghiEXAMPLE%22,%20%22ClientIpAddress%22%3A%22192.0.2.252%22%7D
```

Amazon Cognito は、次の入カイベントをトークン生成前の Lambda トリガーに渡します。

```
{
```

```

version: '3',
triggerSource: 'TokenGeneration_ClientCredentials',
region: 'us-east-1',
userPoolId: 'us-east-1_EXAMPLE',
userName: 'ClientCredentials',
callerContext: {
  awsSdkVersion: 'aws-sdk-unknown-unknown',
  clientId: '1example23456789'
},
request: {
  userAttributes: {},
  groupConfiguration: null,
  scopes: [
    'resourceServerIdentifier1/scope1',
    'resourceServerIdentifier2/scope2'
  ],
  clientMetadata: {
    'onBehalfOfToken': 'eyJra789ghiEXAMPLE',
    'ClientIpAddress': '192.0.2.252'
  }
},
response: { claimsAndScopeOverrideDetails: null }
}

```

レスポンスはアクセストークンを返します。クライアント認証情報の付与は、Machine to Machine (M2M) 認可用であり、アクセストークンのみを返します。

```

HTTP/1.1 200 OK
Content-Type: application/json
{
  "access_token": "eyJra1example",
  "token_type": "Bearer",
  "expires_in": 3600
}

```

POST 本文認可によるクライアント認証情報

次のクライアント認証情報の付与リクエストには、リクエスト本文に `client_secret` パラメータが含まれ、`Authorization` ヘッダーは含まれていません。このリクエストでは、`client_secret_post` 認可構文を使用しています。リクエストにより、リクエストされたスコープを持つアクセストークンが生成されます。リクエストには、IP アドレス情報を提供するク

クライアントメタデータと、付与対象であるユーザーに発行されたトークンも含まれます。Amazon Cognito は、クライアントメタデータをトークン生成前の Lambda トリガーに渡します。

```
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%2Fmy_custom_scope&
client_secret=9example87654321&
aws_client_metadata=%7B%22onBehalfOfToken%22%3A%22eyJra789ghiEXAMPLE%22,%20%22ClientIpAddress%22%3A%22192.0.2.252%22%7D
```

Amazon Cognito は、次の入力イベントをトークン生成前の Lambda トリガーに渡します。

```
{
  version: '3',
  triggerSource: 'TokenGeneration_ClientCredentials',
  region: 'us-east-1',
  userPoolId: 'us-east-1_EXAMPLE',
  userName: 'ClientCredentials',
  callerContext: {
    awsSdkVersion: 'aws-sdk-unknown-unknown',
    clientId: '1example23456789'
  },
  request: {
    userAttributes: {},
    groupConfiguration: null,
    scopes: [
      'resourceServerIdentifier1/my_custom_scope'
    ],
    clientMetadata: {
      'onBehalfOfToken': 'eyJra789ghiEXAMPLE',
      'ClientIpAddress': '192.0.2.252'
    }
  }
}
```

```
  },  
  response: { claimsAndScopeOverrideDetails: null }  
}
```

レスポンスはアクセストークンを返します。クライアント認証情報の付与は、Machine to Machine (M2M) 認可用であり、アクセストークンのみを返します。

```
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"  
}
```

PKCE による認証コード付与

次のリクエスト例では、[PKCE](#) を使用した認証コード付与リクエストに `code_challenge_method` パラメータと `code_challenge` パラメータが含まれた認可リクエストを完了します。

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token  
Content-Type='application/x-www-form-urlencoded'&  
Authorization=Basic ZGpj0Th1M2ppZWRTaTI4M2V1OTI4OmFiY2RLZjAxMjM0NTY3ODkw  
  
grant_type=authorization_code&  
client_id=1example23456789&  
code=AUTHORIZATION_CODE&  
code_verifier=CODE_VERIFIER&  
redirect_uri=com.myclientapp://myclient/redirect
```

レスポンスは、アプリケーションによる正常な PKCE 検証から ID、アクセストークン、更新トークンを返します。

```
HTTP/1.1 200 OK  
Content-Type: application/json  
  
{  
  "access_token": "eyJra1example",  
  "id_token": "eyJra2example",
```

```
{
  "refresh_token": "eyJj3example",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

更新トークンのローテーションなしのトークン更新

次のリクエスト例では、更新トークンの[ローテーション](#)が非アクティブなアプリケーションクライアントに更新トークンを提供します。アプリケーションクライアントにはクライアントシークレットがあるため、リクエストは Authorization ヘッダーを提供します。

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWRTaTI4M2V10TI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=refresh_token&
client_id=1example23456789&
refresh_token=eyJj3example
```

レスポンスは、新しい ID トークンとアクセストークンを返します。

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "eyJra1example",
  "id_token": "eyJra2example",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

更新トークンのローテーションによるトークン更新

次のリクエスト例では、[更新トークンのローテーション](#)がアクティブなアプリケーションクライアントに更新トークンを提供します。アプリケーションクライアントにはクライアントシークレットがあるため、リクエストは Authorization ヘッダーを提供します。

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpj0Th1M2ppZWRTaTI4M2V10TI4OmFiY2RlZjAxMjM0NTY3ODkw
```

```
grant_type=refresh_token&
client_id=1example23456789&
refresh_token=eyJj3example
```

レスポンスは、新しい ID、アクセス、更新トークンを返します。

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "eyJra1example",
  "id_token": "eyJra2example",
  "refresh_token": "eyJj4example",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

否定応答の例

不正な形式のリクエストは、トークンエンドポイントからエラーを生成します。トークンリクエストがエラーを生成したときのレスポンス本文の一般的なマップは、以下のとおりです。

```
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

リクエストに必須パラメータが含まれていない、サポートされていないパラメータ値 (unsupported_grant_type 以外) が含まれている、または正しい形式ではありません。たとえば、grant_type が refresh_token であるが、refresh_token が含まれていない、などです。

invalid_client

クライアントの認証に失敗しました。たとえば、クライアントが認可ヘッダーに client_id と client_secret を含めたが、その client_id と client_secret を持つクライアントが存在しない場合です。

invalid_grant

更新トークンが失効しています。

認可コードが既に消費されているか、存在しません。

アプリクライアントにはリクエストされたスコープ内すべての[属性](#)への読み取りアクセス権はありません。例えば、アプリが email スコープをリクエストすると、アプリクライアントは email 属性の読み取りはできますが、email_verified は読み取れません。

unauthorized_client

クライアントがコード付与フローまたはトークンの更新を許可されていません。

unsupported_grant_type

grant_type が authorization_code、refresh_token、または client_credentials 以外の場合に返されます。

ユーザー属性エンドポイント

OIDC がユーザー属性を含む ID トークンを発行する場合、OAuth 2.0 は /oauth2/userInfo エンドポイントを実装します。認証されたユーザーまたはクライアントは、scopes クレームを含むアクセストークンを受け取ります。このクレームは、認可サーバーが返す属性を決定します。アプリケーションが userInfo エンドポイントにアクセストークンを提示すると、認可サーバーは、アクセストークンスコープによって設定された境界内にあるユーザー属性を含む応答本文を返します。アプリケーションは、少なくとも openid スコープクレームを持つ有効なアクセストークンを保持している限り、userInfo エンドポイントからユーザーに関する情報を取得できます。

userInfo エンドポイントは OpenID Connect (OIDC) [userInfo エンドポイント](#)です。[トークントークンエンドポイント](#)が発行したアクセストークンをサービスプロバイダーが提示すると、ユーザー属性で応答します。ユーザーのアクセストークンのスコープは、userInfo エンドポイントがレスポンスで返すユーザー属性を定義します。openid スコープはアクセストークンクレームのいずれかである必要があります。

Amazon Cognito は、[InitiateAuth](#) などのユーザープール API リクエストに응答してアクセストークンを発行します。スコープが含まれていないため、userInfo エンドポイントはこれらのアクセストークンを受け入れません。代わりに、トークンエンドポイントからアクセストークンを提示する必要があります。

OAuth 2.0 サードパーティーの ID プロバイダー (IdP) は、userInfo エンドポイントもホストしています。ユーザーがその IdP で認証すると、Amazon Cognito は、IdP token エンドポイントと認可コー

ドを表示せずに交換します。ユーザープールは IdP アクセストークンを渡して、IdP userInfo エンドポイントからのユーザー情報の取得を認可します。

ユーザーのアクセストークンのスコープは、認証リクエストの scopes リクエストパラメータ、または[トークン生成前の Lambda トリガー](#)が追加するスコープによって決まります。アクセストークンをデコードし、scope クレームを調べることで、それらに含まれるアクセスコントロールのスコープを確認できます。以下は、userInfo エンドポイントから返されるデータに影響を与えるスコープの組み合わせです。予約済みの Amazon Cognito スコープ `aws.cognito.signin.user.admin` は、このエンドポイントから返されるデータには影響しません。

アクセストークンのスコープの例と **userInfo** レスポンスへの影響

openid

アプリケーションクライアントが読み取れるすべてのユーザー属性を含むレスポンスを返します。

openid profile

ユーザー属性とし

て、name、family_name、given_name、middle_name、nickname、preferred_username、profile を返します。また、[カスタム属性](#)も返します。各属性への読み取りアクセス権を持たないアプリケーションクライアントの場合、このスコープへのレスポンスは、アプリケーションクライアントが読み取りアクセス権を持つ仕様内のすべての属性となります。

openid email

基本的なプロフィール情報と、email 属性および email_verified 属性を返します。

openid phone

基本的なプロフィール情報と、phone_number 属性および phone_number_verified 属性を返します。

GET /oauth2/userInfo

アプリケーションは、ブラウザ経由ではなく、このエンドポイントへのリクエストを直接生成します。

詳細については、OpenID Connect (OIDC) 仕様の「[UserInfo エンドポイント](#)」を参照してください。

トピック

- [ヘッダーのリクエストパラメータ](#)
- [例 – リクエスト](#)
- [例 – 肯定応答](#)
- [否定応答の例](#)

ヘッダーのリクエストパラメータ

Authorization: Bearer *<access_token>*

認可ヘッダーフィールド内でアクセストークンを渡します。

必須。

例 – リクエスト

```
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
```

例 – 肯定応答

```
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"
}
```

OIDC クレームのリストについては、「[スタンダードクレーム](#)」を参照してください。現在、Amazon Cognito は、`email_verified` と `phone_number_verified` の値を文字列として返します。

否定応答の例

例 – 不正なリクエスト

```
HTTP/1.1 400 Bad Request
WWW-Authenticate: error="invalid_request",
error_description="Bad OAuth2 request at UserInfo Endpoint"
```

invalid_request

リクエストに必須パラメータが含まれていないか、サポートされていないパラメータ値が含まれているか、または正しい形式ではありません。

例 – 不正なトークン

```
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

アクセストークンの有効期限が切れているか、失効しているか、正しい形式ではないか、または無効です。

トークン取り消しエンドポイント

セッションで更新トークンを保持するユーザーは、ブラウザ Cookie のようなものが与えられます。更新トークンが有効である限り、既存のセッションを更新できます。ID トークンまたはアクセストークンの有効期限が切れた後にサインインするようにユーザーに促す代わりに、アプリケーションは、更新トークンを使用して新しい有効なトークンを取得できます。ただし、ユーザーのセッションを終了する必要があると外部で判断することや、ユーザーが現在のセッションを忘れたりすることがあります。その時点で、セッションを維持できなくなるように、その更新トークンを取り消せます。

/oauth2/revoke エンドポイントは、指定した更新トークンを使用して Amazon Cognito が最初に発行したユーザーのアクセストークンを取り消します。また、このエンドポイントは、更新トークン自体、および同じ更新トークンからの後続のアクセストークンと ID トークンのすべても取り消します。エンドポイントがトークンを取り消した後、取り消されたトークンを使用して Amazon Cognito トークンが認証する API にアクセスすることはできません。

POST/oauth2/revoke

/oauth2/revoke エンドポイントは HTTPS POST のみをサポートします。ユーザープールクライアントは、システムブラウザ経由ではなくこのエンドポイントに直接リクエストを送信します。

ヘッダーのリクエストパラメータ

Authorization

アプリケーションクライアントにクライアントシークレットがある場合、アプリケーションは、その `client_id` および `client_secret` をベーシック HTTP 認可を介して認可ヘッダーに渡す必要があります。シークレットは [ベーシック](#) `Base64Encode(client_id:client_secret)` です。

Content-Type

常に 'application/x-www-form-urlencoded' にする必要があります。

本文のリクエストパラメータ

token

(必須) クライアントが取り消す対象の更新トークンです。このリクエストは、Amazon Cognito がこの更新トークンで発行したすべてのアクセストークンも無効にします。

必須。

client_id

(オプション) 取り消す対象のトークンのアプリケーションクライアント ID。

クライアントがパブリックでありシークレットがない場合は必須です。

取り消しリクエストの例

この取り消しリクエストは、クライアントシークレットを持たないアプリケーションクライアントの更新トークンを取り消します。リクエスト本文内の client_id パラメータに注意してください。

```
POST /oauth2/revoke HTTP/1.1
Host: mydomain.auth.us-east-1.amazoncognito.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
token=2YotnFZFEjr1zCsicMWpAA&
client_id=1example23456789
```

この取り消しリクエストは、クライアントシークレットを持つアプリケーションクライアントの更新トークンを取り消します。エンコードされたクライアント ID とクライアントシークレットを含むが、リクエスト本文内に client_id を含まない Authorization ヘッダーに注意してください。

```
POST /oauth2/revoke HTTP/1.1
Host: mydomain.auth.us-east-1.amazoncognito.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
token=2YotnFZFEjr1zCsicMWpAA
```

取り消しエラーレスポンス

正常なレスポンスには空のボディが含まれています。エラーレスポンスは、error フィールドを持つ JSON オブジェクトで、error_description フィールドがある場合もあります。

エンドポイントエラー

- リクエストにトークンが存在しない場合、またはアプリケーションクライアントに対して機能が無効化されている場合に、HTTP 400 とエラー invalid_request が返されます。

- Amazon Cognito が取り消しリクエストで送信したトークンが更新トークンでない場合は、HTTP 400 とエラー `unsupported_token_type` が発生します。
- クライアントの認証情報が有効でない場合、HTTP 401 とエラー `invalid_client` が発生します。
- トークンが取り消されている、またはクライアントが無効なトークンを送信した場合は、HTTP 200 OK が発生します。

IdP SAML アサーションエンドポイント

`/saml2/idpresponse` は SAML アサーションを受け取ります。サービスプロバイダーが開始した (SP が開始した) サインインでは、アプリケーションはこのエンドポイントと直接インタラクションを行いません。SAML 2.0 ID プロバイダー (IdP) は、SAML 応答を使用してユーザーをここにリダイレクトします。SP が開始したサインインの場合、`saml2/idpresponse` へのパスをアサーションコンシューマーサービス (ACS) の URL として IdP を設定します。セッション開始の詳細については、「[Amazon Cognito ユーザープールでの SAML セッション開始](#)」を参照してください。

IdP が開始したサインインでは、SAML 2.0 プロバイダーでユーザーをサインインした後に、アプリケーションでこのエンドポイントへのリクエストを呼び出します。ユーザーがブラウザで IdP でサインインすると、アプリケーションは SAML アサーションを収集し、このエンドポイントに送信します。HTTPS 経由で HTTP POST リクエストの本文に SAML アサーションを送信する必要があります。POST リクエストの本文は `SAMLResponse` パラメータと `Relaystate` パラメータである必要があります。詳細については、「[IdP が開始した SAML サインインを実装する](#)」を参照してください。

`saml2/idpresponse` エンドポイントは、最大 100,000 文字長の SAML アサーションを受け入れることができます。

POST `/saml2/idpresponse`

IdP が開始したサインインで `/saml2/idpresponse` エンドポイントを使用するには、ユーザーのセッションに関する情報をユーザープールに提供するパラメータを含む POST リクエストを生成します。

- サインインするアプリケーションクライアント。
- 終了する対象のコールバック URL。
- ユーザーのアクセストークンでリクエストする OAuth 2.0 スコープ。
- サインインリクエストを開始した IdP。

IdP が開始したリクエスト本文のパラメータ

SAMLResponse

ユーザープール内の有効なアプリケーションクライアントと IdP 設定に関連付けられた IdP からの Base64 でエンコードされた SAML アサーション。

RelayState

RelayState パラメータには、oauth2/authorize エンドポイントに本来であれば渡すリクエストパラメータが含まれます。これらのパラメータの詳細については、「[認可エンドポイント](#)」を参照してください。

response_type

OAuth 2.0 付与タイプ。

client_id

アプリケーションクライアント ID。

redirect_uri

Amazon Cognito がユーザーを認証した後、認証サーバーによってブラウザがリダイレクトされる URL です。

identity_provider

ユーザーをリダイレクトする ID プロバイダーの名前。

idp_identifier

ユーザーをリダイレクトする ID プロバイダーの識別子。

scope

ユーザーが認可サーバーにリクエストする OAuth 2.0 スコープ。

肯定応答を含むリクエストの例

例 - POST リクエスト

次のリクエストは、アプリケーションクライアント 1example23456789 の IdP MySAMLIdP からのユーザーに対する認可コードの付与に対するものです。ユーザーは、認可コードを使用し

て `https://www.example.com` にリダイレクトします。認証コードは、OAuth 2.0 スコープ `openid`、`email`、`phone` を持つアクセストークンなどのトークンと交換できます。

```
POST /saml2/idpresponse 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
```

例 – 応答

前のリクエストに対するレスポンス？の例を次に示します。

```
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]
```

OAuth 2.0 許可

Amazon Cognito ユーザープール OAuth 2.0 認可サーバーは、3 種類の OAuth 2.0 [認可付与](#)に対応してトークンを発行します。ユーザープール内の各アプリクライアントに、サポートされる許可タイプを設定できます。暗黙的付与または認可コード付与と同じアプリケーションクライアントで、クライアント認証情報付与を有効にすることはできません。暗黙的コード付与と認可コード付与のリクエストは、[認可エンドポイント](#) で開始され、クライアント認証情報付与のリクエストは [トークンエンドポイント](#) で開始されます。

認可コード付与

認証リクエストが成功すると、認可サーバーは、`code` パラメータの認可コードをコールバック URL のパラメータに追加します。次に、[トークンエンドポイント](#) と、アクセス、ID、更新トークンのコードを交換することができます。認可コード付与をリクエストするには、リクエストで `response_type` を `code` に設定します。リクエスト例については、「[例: 認証コード付与](#)」を参照してください。Amazon Cognito は、認証コード付与で [Proof Key for Code Exchange \(PKCE\)](#) をサポートしています。

認可コード付与は、最も安全な認可付与の形式です。トークンの内容がユーザーに直接表示されることはありません。代わりに、ユーザーのトークンを取得して安全に保管する責任はアプリにあります。Amazon Cognito では、認可コード付与が 3 種類のトークン (ID、アクセス、更新) すべてを認可サーバーから取得する唯一の方法です。Amazon Cognito ユーザープール API による認証から 3 種類のトークンをすべて取得することもできますが、API は `aws.cognito.signin.user.admin` 以外のスコープを持つアクセストークンを発行しません。

暗黙の許可

認証リクエストが成功すると、認可サーバーは `access_token` パラメータのアクセストークンと、`id_token` パラメータの ID トークンを、コールバック URL に追加します。暗黙的な許可では、[トークンエンドポイント](#) に追加の操作は必要ありません。暗黙的な許可をリクエストするには、リクエストで `response_type` を `token` に設定します。暗黙的な許可では、ID とアクセストークンのみが生成されます。リクエスト例については、「[例: openid スコープなしのトークンの \(暗黙的\) 付与](#)」を参照してください。

暗黙的な許可は、レガシーの認可付与方法です。認可コード付与とは異なり、ユーザーはトークンを傍受して検査できます。暗黙的な許可によるトークンの配信を防ぐには、認可コード付与のみをサポートするようにアプリクライアントを設定します。

クライアント認証情報

クライアント認証情報は、マシン間アクセスに対する認可のみを付与します。クライアント認証情報許可を受けるには、[認可エンドポイント](#) をバイパスして、[トークンエンドポイント](#) に直接リクエストを生成してください。アプリクライアントにはクライアントシークレットが必要で、クライアント認証情報許可のみをサポートする必要があります。リクエストが成功すると、認可サーバーがアクセストークンを返します。

クライアント認証情報許可からのアクセストークンは、OAuth 2.0 スコープを含む認可メカニズムです。通常、トークンには、アクセス保護された API に対する HTTP 操作を許可するカスタムスコープクレームが含まれます。詳細については、「[スコープ、M2M、リソースサーバー](#)」を参照してください。

クライアント認証情報付与は、AWS 請求書にコストを追加します。詳細については、「[Amazon Cognito の料金](#)」を参照してください。

更新トークン

更新トークンの付与は、[トークンエンドポイント](#) から直接リクエストできます。この付与は、有効な更新トークンと引き換えに、新しい ID トークンとアクセストークンを返します。

これらの付与とその実装の詳細については、「AWS Security Blog」の「[How to use OAuth 2.0 in Amazon Cognito: Learn about the different OAuth 2.0 grants](#)」を参照してください。

認可コード許可での PKCE の使用

Amazon Cognito は、認可コード付与で Proof Key for Code Exchange (PKCE)をサポートしています。PKCE は、パブリッククライアント向けの OAuth 2.0 認可コード付与の拡張機能です。PKCE は、傍受された認可コードの引き換えから保護します。

Amazon Cognito での PKCE の使用方法

PKCE で認証を開始するには、アプリケーションが一意の文字列値を生成する必要があります。この文字列は、最初の認可付与をリクエストするクライアントと、トークンの認可コードを交換するクライアントとを比較するために Amazon Cognito が使用するシークレット値であるコード検証子です。

アプリケーションは、コード検証文字列に SHA256 ハッシュを適用し、結果を base64 にエンコードする必要があります。ハッシュされた文字列をリクエスト本文の code_challenge パラメータとして [認可エンドポイント](#) に渡します。アプリケーションが認可コードとトークンを交換する場合、[トークンエンドポイント](#) へのリクエスト本文に code_verifier パラメータとしてプレーンテキストのコード検証文字列を含める必要があります。Amazon Cognito は、コード検証子に対して同じハッシュアンドエンコードオペレーションを実行します。Amazon Cognito は、コード検証子が認可リクエストで受け取ったものと同じコードチャレンジをもたらすと判断した場合のみ、ID トークン、アクセストークン、および更新トークンを返します。

PKCE を使用して認可付与フローを実装するには

1. [Amazon Cognito コンソール](#)を開きます。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択、または[ユーザープールを作成](#)します。ユーザープールを作成する場合は、ウィザード中にアプリケーションクライアントのセットアップとマネージドログインの設定を求められます。
 - a. 新しいユーザープールを作成する場合は、ガイド付きセットアップ中にアプリケーションクライアントをセットアップし、マネージドログインを設定します。
 - b. 既存のユーザープールを設定する場合で、[ドメイン](#)と[パブリックアプリケーションクライアント](#)がない場合は、これらを追加します。

4. PKCE のコードチャレンジを作成するには、ランダムな英数字文字列、通常は Universally Unique Identifier ([UUID](#)) を生成します。この文字列は、リクエストで [トークンエンドポイント](#) に送信する code_verifier パラメータの値です。
5. SHA256 アルゴリズムを使用して code_verifier 文字列をハッシュします。ハッシュ操作の結果を base64 にエンコードします。この文字列は、リクエストで [認可エンドポイント](#) に送信する code_challenge パラメータの値です。

次の Python 例では、code_verifier を生成し、code_challenge を計算します。

```
#!/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}")
```

Python スクリプトからの出力例を次に示します。

```
code challenge: Eh0mg-0Zv7BAyo-tdv_vYamx1bo0YDulDklyXoMDtLg
code verifier: 9D-aW_iygXrgQcWJd0y0tNVMPsXSchIc2xceDhvYVdGLCBk-
JWFTmBNjvKSd0rjTTYaz0FbUmrFERrjWx6oKtK2b6z_x4_gHBDlr4K1mRFgyE8yA-05-_v7Dxf3EIYJH
```

6. PKCE を使用した認証コード付与リクエストで、マネージドログインのサインインを完了します。URL の例を次に示します。

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com&code_challenge=Eh0mg-0Zv7BAyo-
tdv_vYamx1bo0YDulDklyXoMDtLg&code_challenge_method=S256
```

7. 認可 code を収集し、トークンエンドポイントを持つトークンと引き換えます。リクエストの例を次に示します。

```
POST /oauth2/token HTTP/1.1
Host: mydomain.auth.us-east-1.amazoncognito.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_iygXrgQcWJd0y0tNVMPsXSchIc2xceDhvYVdGLCBk-
JWFTmBNjvKSd0rjTTYaz0FbUmrFERrjWx6oKtK2b6z_x4_gHBDlr4K1mRFGyE8yA-05-_v7Dxf3EIYJH
```

8. 応答を確認します。ID トークン、アクセストークン、更新トークンが含まれます。Amazon Cognito ユーザープールトークン使用の詳細については、「[ユーザープール JSON ウェブトークン \(JWT\) の理解](#)」を参照してください。

マネージドログインとフェデレーションのエラーレスポンス

マネージドログインまたはフェデレーティッドサインインからエラーが返されることがあります。認証がエラーで終了する原因となる条件は次のとおりです。

- ユーザープールでは実行できない操作を、ユーザーが実行する。
- Lambda トリガーが想定どおりの構文で応答しない。
- ID プロバイダー (IdP) がエラーを返す。
- Amazon Cognito は、ユーザーが提供した属性情報を検証できませんでした。
- IdP は、必須の属性に対応するクレームを送信しませんでした。

Amazon Cognito はエラーを検出すると、以下のいずれかの方法でエラーを通知します。

1. Amazon Cognito は、リクエストパラメータにエラーを含むリダイレクト URL を送信します。
2. Amazon Cognito はマネージドログインのエラーを表示します。

Amazon Cognito がリクエストパラメータに追加するエラーの形式は次のとおりです。

```
https://<Callback URL>/?error_description=error+description&error=error+name
```

ユーザーが操作を実行できないときにエラー情報を送信できるようにする場合は、URL とテキスト、またはページのスクリーンショットをキャプチャするようにリクエストしてください。

Note

Amazon Cognito エラーの説明は固定文字列ではないため、固定のパターンや形式に依存するロジックは使用しないでください。

OIDC とソーシャル ID プロバイダーのエラーメッセージ

ID プロバイダー (IdP) は、エラーを返すことがあります。OIDC または OAuth 2.0 IdP が標準に準拠したエラーを返すと、Amazon Cognito はユーザーをコールバック URL にリダイレクトし、プロバイダーのエラーレスポンスをエラーリクエストパラメータに追加します。Amazon Cognito は、既存のエラー文字列にプロバイダー名と HTTP エラーコードを追加します。

次の URL は、エラーを返した IdP から Amazon Cognito へのリダイレクトの例です。

```
https://www.amazon.com/?error_description=LoginWithAmazon+Error+-+400+invalid_request+The+request+is+missing+a+required+parameter+%3A+client_secret&error=invalid_request
```

Amazon Cognito はプロバイダーから受け取ったもののみを返すため、ユーザーにはこの情報のサブセットが表示される場合があります。

IdP による初回ログインで問題が発生すると、IdP はすべてのエラーメッセージをユーザーに直接配信します。Amazon Cognito は、ユーザーセッションを検証するリクエストを IdP に生成すると、エラーメッセージをユーザーに中継します。Amazon Cognito は、以下のエンドポイントからの OAuth および OIDC IdP のエラーメッセージを中継します。

/token

Amazon Cognito が IdP の認可コードをアクセストークンと交換します。

/.well-known/openid-configuration

Amazon Cognito は発行者のエンドポイントへのパスを検出します。

`/.well-known/jwks.json`

ユーザーの JSON ウェブトークン (JWT) を検証するために、Amazon Cognito は IdP がトークンの署名に使用する JSON ウェブキー (JWK) を検出します。

Amazon Cognito は HTTP エラーを返す可能性のある SAML 2.0 プロバイダーへのアウトバウンドセッションを開始しないため、SAML 2.0 IdP とのセッション中のユーザーエラーには、この形式のプロバイダーエラーメッセージは含まれません。

Amazon Cognito アイデンティティプール

Amazon Cognito アイデンティティプールは、AWS 認証情報と交換できるフェデレーションアイデンティティのディレクトリです。ID プールは、サインインしているか、まだ識別していないかにかかわらず、アプリケーションのユーザーに一時的な AWS 認証情報を生成します。AWS Identity and Access Management (IAM) ロールとポリシーを使用すると、ユーザーに付与するアクセス許可のレベルを選択できます。ユーザーはゲストとしてスタートし、AWS のサービスで保管しているアセットを取得できます。その後、サードパーティー ID プロバイダーにサインインして、登録メンバーに提供したアセットへのアクセスをロック解除できます。サードパーティーの ID プロバイダーには、Apple や Google などのコンシューマー (ソーシャル) OAuth 2.0 プロバイダー、カスタム SAML または OIDC ID プロバイダー、またはお客様が独自に設計したカスタム認証スキーム (開発者プロバイダーとも呼ばれます) があります。

Amazon Cognito アイデンティティプールの機能

の署名リクエスト AWS のサービス

Amazon Simple Storage Service (Amazon S3) や Amazon DynamoDB AWS のサービスなどの [API リクエストに署名](#)します。Amazon Pinpoint や Amazon CloudWatch などのサービスを使用してユーザーアクティビティを分析します。

リソースベースのポリシーを使用してリクエストをフィルタリングする

リソースへのユーザーアクセスをきめ細かに制御します。ユーザークレームを [IAM セッションタグ](#)に変換し、ユーザーの個別のサブセットにリソースアクセスを許可する IAM ポリシーを構築します。

ゲストアクセスを割り当てる

まだサインインしていないユーザーの場合は、アクセス範囲の狭い AWS 認証情報を生成するようにアイデンティティプールを設定します。シングルサインオンプロバイダーを通じてユーザーを認証し、アクセスを強化します。

ユーザーの特性に基づいて IAM ロールを割り当てる

認証されたすべてのユーザーに 1 つの IAM ロールを割り当てるか、各ユーザーのリクエストに基づいてロールを選択します。

さまざまな ID プロバイダーを受け入れる

ID またはアクセストークン、ユーザープールトークン、SAML アサーション、またはソーシャルプロバイダーの OAuth トークンを AWS 認証情報と交換します。

自分のアイデンティティを検証する

独自のユーザー検証を実行し、開発者 AWS 認証情報を使用してユーザーの認証情報を発行します。

アプリに認証および認可サービスを提供する Amazon Cognito ユーザープールがすでにある場合があります。ユーザープールは ID プロバイダー (IdP) としてユーザープールに設定できます。これを行うと、ユーザーはユーザープール IdPs を介して認証し、クレームを共通の OIDC ID トークンに統合して、そのトークンを AWS 認証情報と交換できます。その後、ユーザーは署名付きのリクエストで認証情報を AWS のサービスに提示できます。

また、任意の ID プロバイダーからの認証済みクレームをアイデンティティプールに直接提示することもできます。Amazon Cognito は、SAML、OAuth、OIDC プロバイダーからのユーザークレームを、短期認証情報の [AssumeRoleWithWebIdentity API](#) リクエストにカスタマイズします。

Amazon Cognito ユーザープールは、SSO 対応アプリケーションの OIDC ID プロバイダーのようなものです。アイデンティティプールは、IAM 認可に最も適したリソース依存関係を持つすべてのアプリケーションの AWS ID プロバイダーとして機能します。

Amazon Cognito ID プールは、以下の ID プロバイダーをサポートします。

- パブリックプロバイダー: [アイデンティティプール IdP として Login with Amazon を設定する](#)、[アイデンティティプール IdP として Facebook を設定する](#)、[アイデンティティプール IdP として Google を設定する](#)、[アイデンティティプール IdP として Apple でサインインを設定する](#)、Twitter。
- [Amazon Cognito ユーザープール](#)
- [OIDC プロバイダーをアイデンティティプール IdP として設定する](#)
- [SAML プロバイダーをアイデンティティプール IdP として設定する](#)
- [デベロッパーが認証した ID](#)

Amazon Cognito ID プールを利用できるリージョンの詳細については、「[AWS リージョン別のサービス](#)」を参照してください。

Amazon Cognito ID プールの詳細については、以下のトピックを参照してください。

トピック

- [アイデンティティプールコンソールの概要](#)

- [ID プールの認証フロー](#)
- [IAM ロール](#)
- [Amazon Cognito アイデンティティプールに対するセキュリティのベストプラクティス](#)
- [アクセスコントロールへの属性の使用](#)
- [ロールベースアクセスコントロールの使用](#)
- [認証情報の取得](#)
- [一時的な認証情報 AWS のサービス を使用した へのアクセス](#)
- [アイデンティティプールのサードパーティー ID プロバイダー](#)
- [デベロッパーが認証した ID](#)
- [認証されていないユーザーから認証されたユーザーへの切り替え](#)

アイデンティティプールコンソールの概要

Amazon Cognito ID プールは、ゲストであるユーザー (認証されていないユーザー) と、認証されてトークンを受け取ったユーザーに一時的な AWS 認証情報を提供します。アイデンティティプールは、外部 ID プロバイダーにリンクされたユーザー識別子のストアです。

アイデンティティプールの機能とオプションを理解する方法の 1 つは、Amazon Cognito コンソールで作成することです。認証フロー、ロールベースおよび属性ベースのアクセスコントロール、ゲストアクセスに対するさまざまな設定の影響を調べることができます。そこから、このガイドの後半の章に進み、適切なコンポーネントをアプリケーションに追加して、アイデンティティプール認証を実装できます。

トピック

- [アイデンティティプールを作成する](#)
- [ユーザー IAM ロール](#)
- [認証された ID と認証されていない ID](#)
- [ゲストアクセスをアクティブ化/非アクティブ化する](#)
- [ID の種類に関連付けられたロールを変更する](#)
- [ID プロバイダーを編集する](#)
- [アイデンティティプールを削除する](#)
- [ID プールから ID を削除する](#)
- [Amazon Cognito Sync を ID プールで使用する](#)

アイデンティティプールを作成する

コンソールで新しい ID プールを作成する

1. [Amazon Cognito コンソール](#)にサインインし、アイデンティティプールを選択します。
2. [ID プールを作成] を選択します。
3. [ID プールの信頼を設定] で、アイデンティティプールを認証アクセス、ゲストアクセス、またはその両方に設定することを選択します。
 - [認証アクセス] を選択した場合、アイデンティティプールの認証済み ID のソースとして設定する ID タイプを 1 つ以上選択します。[カスタムデベロッパープロバイダー] を設定した場合、アイデンティティプールの作成後にそのプロバイダーを変更したり削除したりすることはできません。
4. [許可を設定] で、アイデンティティプール内の認証済みユーザーまたはゲストユーザーのデフォルトの IAM ロールを選択します。
 - a. Amazon Cognito に、基本的な権限とアイデンティティプールとの信頼関係を持つ新しいロールを作成する場合は、[新しい IAM ロールを作成] を選択します。新しいロールを識別するための IAM ロール名を入力します (たとえば myidentitypool_authenticatedrole)。[ポリシードキュメントを表示] を選択して、Amazon Cognito が新しい IAM ロールに割り当てるアクセス権限を確認します。
 - b. 使用する に既にロールがある場合は、既存の IAM ロール AWS アカウント を使用することを選択できます。cognito-identity.amazonaws.com を含めるように IAM ロールの信頼ポリシーを設定する必要があります。リクエストが特定のアイデンティティプール内の認証されたユーザーから送信されたという証拠を提示した場合にのみ、Amazon Cognito がロールを引き継ぐことを許可するようにロールの信頼ポリシーを設定します。詳細については、「[ロールの信頼とアクセス権限](#)」を参照してください。
5. [ID プロバイダーを接続] で、[ID プールの信頼の設定] で選択した ID プロバイダー (IdPs) の詳細を入力します。OAuth アプリケーションクライアント情報の提供、Amazon Cognito ユーザープールの選択、IAM IdP の選択、またはデベロッパープロバイダーのカスタム識別子の入力を求められる場合があります。
 - a. 各 IdP の [ロール設定] を選択します。その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。Amazon Cognito ユーザープール IdP では、トークンに preferred_role を含むロールを選択することもできます。cognito:preferred_role クレームの詳細については、「[グループへの優先順位の値の割り当て](#)」を参照してください。

- i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
 - b. アクセスコントロールの属性は、IdP ごとに個別に設定できます。アクセスコントロールの属性は、Amazon Cognito がユーザーの一時セッションに適用する [プリンシパルタグ](#) にユーザーのクレームをマッピングします。セッションに適用するタグに基づいてユーザーアクセスをフィルタリングする IAM ポリシーを作成できます。
 - i. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - ii. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - iii. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
6. [プロパティの設定] の [ID プール名] に [名前] を入力します。
 7. [基本 (クラシック) 認証] で、ベーシックフローを有効にするかどうかを選択します。基本フローがアクティブになっていると、IdPs に対して行ったロール選択をバイパスして [AssumeRoleWithWebIdentity](#) を直接呼び出すことができます。詳細については、「[ID プールの認証フロー](#)」を参照してください。
 8. アイデンティティプールに [タグ](#) を適用する場合は、[タグ] で [タグの追加] を選択します。
 9. [確認および作成] で、新しいアイデンティティプールに対して行った選択を確認します。[編集] を選択してウィザードに戻り、設定を変更します。終了したら、[ID プールの作成] を選択します。

ユーザー IAM ロール

IAM ロールは、ユーザーがなどの AWS リソースにアクセスするためのアクセス許可を定義します [Amazon Cognito Sync](#)。アプリケーションのユーザーは、作成されたロールを引き受けます。認証されたユーザーと認証されていないユーザー用に、異なるロールを指定できます。IAM ロールの詳細については、「[IAM ロール](#)」を参照してください。

認証された ID と認証されていない ID

Amazon Cognito ID プールは、認証された ID と認証されていない ID の両方をサポートします。認証された ID は任意のサポートされている認証プロバイダーで認証されたユーザーに属します。通常、認証されていない ID はゲストユーザーに属します。

- パブリックログインプロバイダーに対して、認証された ID を設定する方法については、「[アイデンティティプールのサードパーティー ID プロバイダー](#)」を参照してください。
- 独自のバックエンド認証プロセスを設定するには、「[デベロッパーが認証した ID](#)」を参照してください。

ゲストアクセスをアクティブ化/非アクティブ化する

Amazon Cognito ID プールのゲストアクセス (認証されていない ID) は、ID プロバイダーで認証しないユーザーに一意の識別子と AWS 認証情報を提供します。アプリケーションで、ログインしないユーザーを許可している場合、認証されていない ID 用にアクセスを許可できます。詳細については[Amazon Cognito アイデンティティプールの開始方法](#)を参照してください。

アイデンティティプールのゲストアクセスを更新するには

- [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
 - [ユーザーアクセス] タブを選択します。
 - ゲストアクセスを検索します。現在ゲストアクセスをサポートしていないアイデンティティプールでは、ステータスが非アクティブです。
 - ゲストアクセスがアクティブで、非アクティブ化したい場合は、[非アクティブ化] を選択します。
 - ゲストアクセスが非アクティブで、アクティブにしたい場合は、[編集] を選択します。
- アイデンティティプールのゲストユーザーのデフォルトの IAM ロールを選択します。
 - Amazon Cognito に、基本的な権限とアイデンティティプールとの信頼関係を持つ新しいロールを作成する場合は、[新しい IAM ロールを作成] を選択します。新しいロールを識別するための IAM ロール名を入力します (たとえば myidentitypool_authenticatedrole)。[ポリシードキュメントを表示] を選択して、Amazon Cognito が新しい IAM ロールに割り当てるアクセス権限を確認します。

- B. 使用する に既にロールがある場合は、既存の IAM ロール AWS アカウント を使用することを選択できます。cognito-identity.amazonaws.com を含めるように IAM ロールの信頼ポリシーを設定する必要があります。リクエストが特定のアイデンティティプール内の認証されたユーザーから送信されたという証拠を提示した場合にのみ、Amazon Cognito がロールを引き継ぐことを許可するようにロールの信頼ポリシーを設定します。詳細については、「[ロールの信頼とアクセス権限](#)」を参照してください。
- C. [変更を保存] を選択します。
- D. ゲストアクセスを有効にするには、[ユーザーアクセス] タブで [アクティブ化] を選択します。

ID の種類に関連付けられたロールを変更する

ID プールのすべての ID は、認証されているか、認証されていないかのいずれかです。認証された ID は、パブリックログインプロバイダー (Amazon Cognito ユーザープール、Login with Amazon、「Apple でサインイン」、Facebook、Google、SAML、または任意の OpenID Connect プロバイダー)、またはデベロッパープロバイダー (独自のバックエンド認証プロセス) によって認証されたユーザーに属します。通常、認証されていない ID はゲストユーザーに属します。

各 ID の種類について、割り当てられたロールがあります。このロールには、AWS のサービス そのロールがアクセスできるロールを指定するポリシーがアタッチされています。リクエストを受け取った Amazon Cognito は、ID タイプとその ID タイプに割り当てられたロールを判別し、そのロールにアタッチされたポリシーを使用して応答を返します。ポリシーを変更するか、アイデンティティタイプに別のロールを割り当てることで、アクセスできるアイデンティティタイプ AWS のサービスを制御できます。ID プールのロールに関連付けられたポリシーの表示または変更については、「[AWS IAM コンソール](#)」を参照してください。

アイデンティティプールのデフォルトの認証済みロールまたは非認証ロールを変更するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ゲストアクセス] または [認証されたアクセス] を見つけます。現在そのアクセスタイプに設定されていないアイデンティティプールでは、ステータスが非アクティブです。[Edit] (編集) を選択します。

4. アイデンティティプールのゲストまたは認証されたユーザーのデフォルトの IAM ロールを選択します。
 - a. Amazon Cognito に、基本的な権限とアイデンティティプールとの信頼関係を持つ新しいロールを作成する場合は、[新しい IAM ロールを作成] を選択します。新しいロールを識別するための IAM ロール名を入力します (たとえば myidentitypool_authenticatedrole)。[ポリシードキュメントを表示] を選択して、Amazon Cognito が新しい IAM ロールに割り当てるアクセス権限を確認します。
 - b. 使用する に既にロールがある場合は、既存の IAM ロール AWS アカウント を使用することを選択できます。cognito-identity.amazonaws.com を含めるように IAM ロールの信頼ポリシーを設定する必要があります。リクエストが特定のアイデンティティプール内の認証されたユーザーから送信されたという証拠を提示した場合にのみ、Amazon Cognito がロールを引き継ぐことを許可するようにロールの信頼ポリシーを設定します。詳細については、「[ロールの信頼とアクセス権限](#)」を参照してください。
5. [変更を保存] を選択します。

ID プロバイダーを編集する

コンシューマー ID プロバイダー (Amazon Cognito ユーザープール、Login with Amazon、Apple でサインイン、Facebook、Google など) を使用した認証をユーザーに許可している場合は、Amazon Cognito ID プール (フェデレーテッドアイデンティティ) のコンソールでアプリケーション識別子を指定できます。これにより、アプリケーション ID (パブリックログインプロバイダーによって提供) が ID プールに関連付けられます。

このページからプロバイダーごとの認証ルールを設定することもできます。プロバイダーごとに、最大 25 個のルールを割り当てることができます。ルールは、プロバイダーごとに保存した順序で適用されます。詳細については、「[ルールベースアクセスコントロールの使用](#)」を参照してください。

Warning

アイデンティティプールでリンクされた IdP アプリケーション ID を変更すると、そのアイデンティティプールによる既存ユーザーの認証ができなくなります。詳細については、「[アイデンティティプールのサードパーティー ID プロバイダー](#)」を参照してください。

アイデンティティプール ID プロバイダー (IdP) を更新するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. ID プロバイダーを見つけます。編集する ID プロバイダーを選択します。新しい IdP を追加する場合は、[ID プロバイダーを追加] を選択します。
 - [ID プロバイダーを追加] を選択した場合は、追加する ID タイプのいずれかを選択します。
4. アプリケーション ID を変更するには、[アイデンティティプロバイダーに関する情報] で [編集] を選択します。
5. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを変更するには、[ロール設定] で [編集] を選択します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。Amazon Cognito ユーザープール IdP では、トークンに `preferred_role` を含むロールを選択することもできます。cognito:preferred_role クレームの詳細については、「[グループへの優先順位の値の割り当て](#)」を参照してください。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] で [編集] を選択します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
7. [変更を保存] を選択します。

アイデンティティプールを削除する

アイデンティティプールの削除は元に戻すことができません。アイデンティティプールを削除すると、そのプールに依存するすべてのアプリとユーザーは機能しなくなります。

ID プールを削除するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。削除するアイデンティティプールの横にあるラジオボタンをオンにします。
2. [削除] を選択します。
3. アイデンティティプールの名前を入力または貼り付けて、[削除] を選択します。

Warning

[Delete] (削除) ボタンをクリックすると、ID プールと、それに含まれているすべてのユーザーデータが完全に削除されます。アイデンティティプールを削除すると、そのアイデンティティプールを使用しているアプリケーションや他のサービスは動作を停止します。

ID プールから ID を削除する

アイデンティティプールから ID を削除すると、Amazon Cognito がそのフェデレーテッドユーザーについて保存した識別情報が削除されます。ユーザーが再度認証情報をリクエストすると、アイデンティティプールが依然として ID プロバイダーを信頼していれば、そのユーザーは新しい ID を受け取ります。このオペレーションは元に戻すことができません。

ID を削除するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ID ブラウザー] タブを選択します。
3. 削除する ID プロバイダーの横にあるチェックボックスを選択してから、[削除] を選択します。ID を削除することを確認し、[削除] を選択します。

Amazon Cognito Sync を ID プールで使用する

Amazon Cognito Sync は、デバイス間でアプリケーション関連のユーザーデータを同期できるようにする AWS のサービス および クライアントライブラリです。Amazon Cognito Sync は、独自のバックエンドを使用せずに、モバイルデバイスとウェブ間でユーザープロフィールデータを同期できます。クライアントライブラリはローカルにデータをキャッシュするため、アプリはデバイスの接続状態にかかわらず、データを読み書きすることができます。デバイスがオンラインになると、データを同期できます。プッシュ同期を設定すると、更新が利用できることが他のデバイスにすぐに通知されます。

データセットの管理

アプリケーションに Amazon Cognito Sync 機能を実装している場合、Amazon Cognito ID プールコンソールで、個々のアイデンティティ用のデータセットとレコードを手動で作成したり、削除したりできます。Amazon Cognito ID プールコンソールで、アイデンティティ用のデータセットまたはレコードに対して行った変更は、コンソールの [Synchronize] (同期) をクリックするまで保存されません。この変更は、Synchronize が呼び出され、ID が同期されるでは、エンドユーザーに表示されません。特定のアイデンティティのデータセット一覧ページを更新すると、個別のアイデンティティについて他のデバイスから同期されたデータが表示されます。

アイデンティティのデータセットを作成する

Amazon Cognito Sync は、データセットを 1 つの ID に関連付けます。ID が表すユーザーに関する識別情報をデータセットに入力し、その情報をユーザーのすべてのデバイスに同期できます。

データセットとデータセットレコードを ID に追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ID ブラウザー] タブを選択します。
3. 編集する ID を選択します。
4. [データセット] で、[データセットの作成] を選択します。
5. データセット名を入力し、[データセットの作成] を選択します。
6. データセットにレコードを追加する場合は、ID の詳細からデータセットを選択します。[レコード] で、[レコードの作成] を選択します。
7. レコードの [キー] と [値] を入力します。[確認] を選択します。同じ手順を繰り返して、さらにレコードを追加します。

ID に関連付けられたデータセットを削除する

ID からデータセットとそのレコードを削除するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ID ブラウザー] タブを選択します。
3. 削除するデータセットを含む ID を選択します。
4. [データセット] で、削除するデータセットの横にあるラジオボタンを選択します。
5. [削除] を選択します。選択内容を確認して、もう一度 [削除] を選択します。

データの一括発行

一括発行を使用すると、既に Amazon Cognito Sync ストアに保存されているデータを、Amazon Kinesis ストリームにエクスポートできます。すべてのストリームを一括発行する手順については、「[Amazon Cognito Sync ストリームの実装](#)」を参照してください。

プッシュ同期を有効にする

Amazon Cognito は自動的にアイデンティティとデバイス間の関係を追跡します。プッシュ同期機能を使用すると、ID データが変更されたときに、特定の ID のすべてのインスタンスに確実に通知されます。プッシュ同期は、ID のデータセットが変更されるたびに、その ID に関連付けられているすべてのデバイスが、この変更を伝えるサイレントプッシュ通知を受け取ることを確実にします。

プッシュ同期は Amazon Cognito コンソールでアクティブ化できます。

プッシュ同期をアクティブ化するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザープールのプロパティ] タブを選択します。
3. [プッシュ同期] で [編集] を選択します
4. [ID プールとのプッシュ同期を有効にする] を選択します。
5. 現在 AWS リージョン で作成した Amazon Simple Notification Service (Amazon SNS) プラットフォームアプリケーションから 1 つを選択します。Amazon Cognito はプッシュ通知をプラットフォームアプリケーションに公開します。[プラットフォームアプリケーションの作成] を選択して Amazon SNS コンソールに移動し、新しいコンソールを作成します。

6. プラットフォームアプリケーションにパブリッシュするには、Amazon Cognito が AWS アカウント で IAM ロールを引き受けます。Amazon Cognito に、基本的な権限とアイデンティティプールとの信頼関係を持つ新しいロールを作成する場合は、[新しい IAM ロールを作成] を選択します。新しいロールを識別するための IAM ロール名を入力します (たとえば `myidentitypool_authenticatedrole`)。[ポリシードキュメントを表示] を選択して、Amazon Cognito が新しい IAM ロールに割り当てるアクセス権限を確認します。
7. 使用する に既にロールがある場合は、既存の IAM ロール AWS アカウント を使用することを選択できます。cognito-identity.amazonaws.com を含めるように IAM ロールの信頼ポリシーを設定する必要があります。リクエストが特定のアイデンティティプール内の認証されたユーザーから送信されたという証拠を提示した場合にのみ、Amazon Cognito がロールを引き継ぐことを許可するようにロールの信頼ポリシーを設定します。詳細については、「[ロールの信頼とアクセス権限](#)」を参照してください。
8. [変更を保存] を選択します。

Amazon Cognito Streams をセットアップする

Amazon Cognito Streams は、Amazon Cognito Sync に保存されているデータに対する制御と洞察をデベロッパーに提供します。これで、デベロッパーはイベントをデータとして受け取るために Kinesis ストリームを設定できるようになります。Amazon Cognito は、所有する Kinesis ストリームに各データセットの変更をリアルタイムでプッシュできます。Amazon Cognito コンソールで Amazon Cognito Streams をセットアップする方法の手順については、「[Amazon Cognito Sync ストリームの実装](#)」を参照してください。

Amazon Cognito Events をセットアップする

Amazon Cognito Events では、Amazon Cognito Sync の重要なイベントに応じて AWS Lambda 関数を実行できます。Amazon Cognito Sync は、データセットが同期されるときに Sync Trigger イベントを生成します。Sync Trigger イベントを使用して、ユーザーがデータを更新するときにアクションを実行できます。コンソールから Amazon Cognito Events をセットアップする手順については、「[Amazon Cognito イベントによるワークフローのカスタマイズ](#)」を参照してください。

詳細については AWS Lambda、「」を参照してください[AWS Lambda](#)。

ID プールの認証フロー

Amazon Cognito は、エンドユーザーのために、デバイスおよびプラットフォーム全体で整合性が維持される一意の識別子を作成するために役立ちます。Amazon Cognito は、権限が制限された一

時的な認証情報をアプリケーションに配信して、AWS リソースにアクセスします。このページでは、Amazon Cognito での認証の基礎と、ID プール内のアイデンティティのライフサイクルについて説明します。

外部プロバイダーの認証フロー

Amazon Cognito で認証されるユーザーは、その認証情報をブートストラップするために複数ステップのプロセスを経由します。Amazon Cognito には、パブリックプロバイダーでの認証に、拡張認証と基本認証の 2 つの異なるフローがあります。

これらのフローのいずれかを完了すると、ロールのアクセスポリシーで定義されている AWS のサービス 他 の にアクセスできます。デフォルトでは、[Amazon Cognito コンソール](#)は、Amazon Cognito Sync ストアと Amazon Mobile Analytics へのアクセス権を持つロールを作成します。その他のアクセス権を付与する方法の詳細については、「[IAM ロール](#)」を参照してください。

アイデンティティプールは、プロバイダーから次のアーティファクトを受け入れます。

プロバイダー	認証アーティファクト
Amazon Cognito ユーザープール	ID トークン
OpenID Connect (OIDC)	ID トークン
SAML 2.0	SAML アサーション
ソーシャルプロバイダー	アクセストークン

拡張 (簡易) 認証フロー

拡張認証フローを使用する場合、アプリケーションは最初に、[GetId](#) リクエストで、認可された Amazon Cognito ユーザープールまたはサードパーティー ID プロバイダーからの認証の証拠を示します。

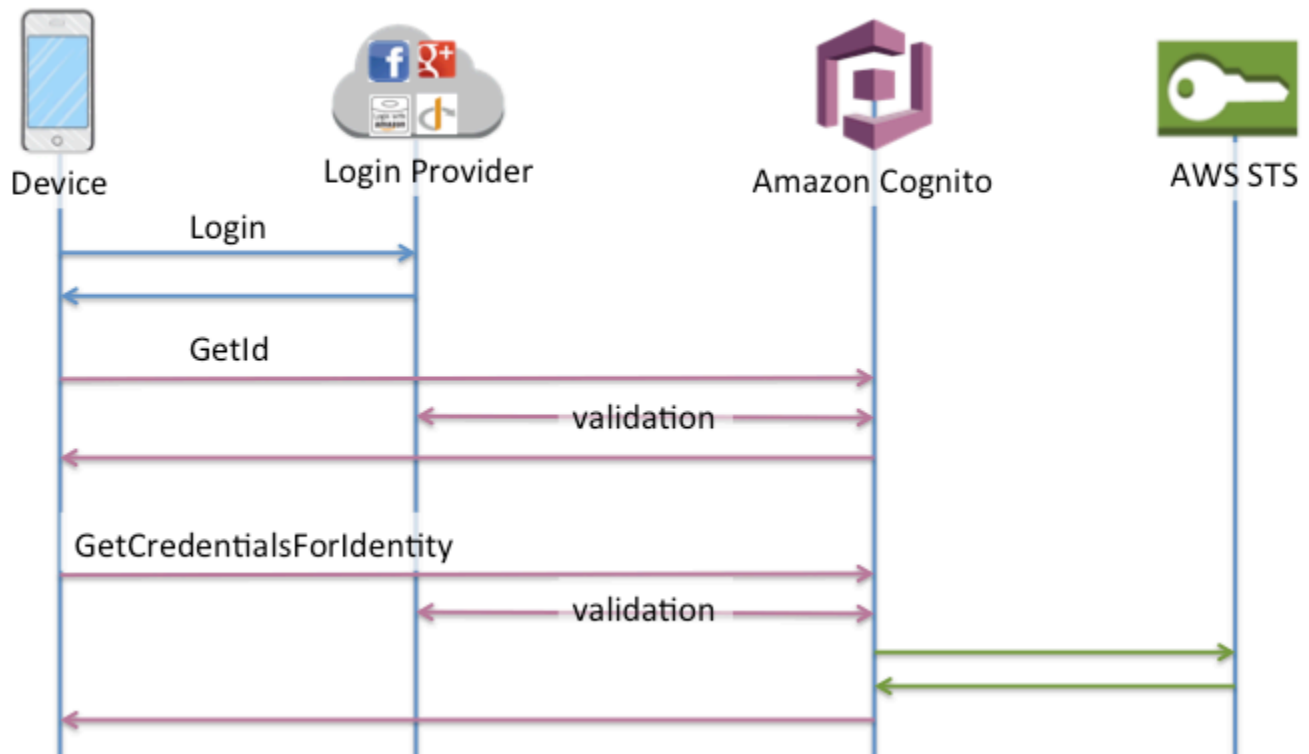
1. アプリケーションは、[GetId](#) リクエストで、認可された Amazon Cognito ユーザープールまたはサードパーティー ID プロバイダーからの JSON ウェブトークンまたは SAML アサーションの認証の証拠を提示します。
2. アイデンティティプールはアイデンティティ ID を返します。
3. アプリケーションは、[GetCredentialsForIdentity](#) リクエストでアイデンティティ ID を同じ認証の証拠と組み合わせます。

4. ID プールは AWS 認証情報を返します。
5. アプリケーションは、一時的な認証情報を使用して AWS API リクエストに署名します。

拡張認証は、アイデンティティプール設定での IAM ロールの選択と認証情報の取得のロジックを管理します。デフォルトのロールを選択するようにアイデンティティプールを設定して、属性ベースのアクセス制御 (ABAC) またはロールベースのアクセス制御 (RBAC) の原則をロール選択に適用できます。拡張認証の AWS 認証情報は 1 時間有効です。

拡張認証でのオペレーションの順序

1. GetId
2. GetCredentialsForIdentity



基本 (クラシック) 認証フロー

基本認証フローを実装すると、アプリケーションは、ユーザーが引き受ける IAM ロールを選択します。

1. アプリケーションは、[GetID](#) リクエストで、認可された Amazon Cognito ユーザープールまたはサードパーティー ID プロバイダーからの JSON ウェブトークンまたは SAML アサーションの認証の証拠を提示します。
2. アイデンティティプールはアイデンティティ ID を返します。
3. アプリケーションは、[GetOpenIdToken](#) リクエストでアイデンティティ ID を同じ認証の証拠と組み合わせます。
4. GetOpenIdToken は、アイデンティティプールによって発行された新しい OAuth 2.0 トークンを返します。
5. アプリケーションは、[AssumeRoleWithWebIdentity](#) リクエストで新しいトークンを提示します。
6. AWS Security Token Service (AWS STS) は AWS 認証情報を返します。
7. アプリケーションは、一時的な認証情報を使用して AWS API リクエストに署名します。

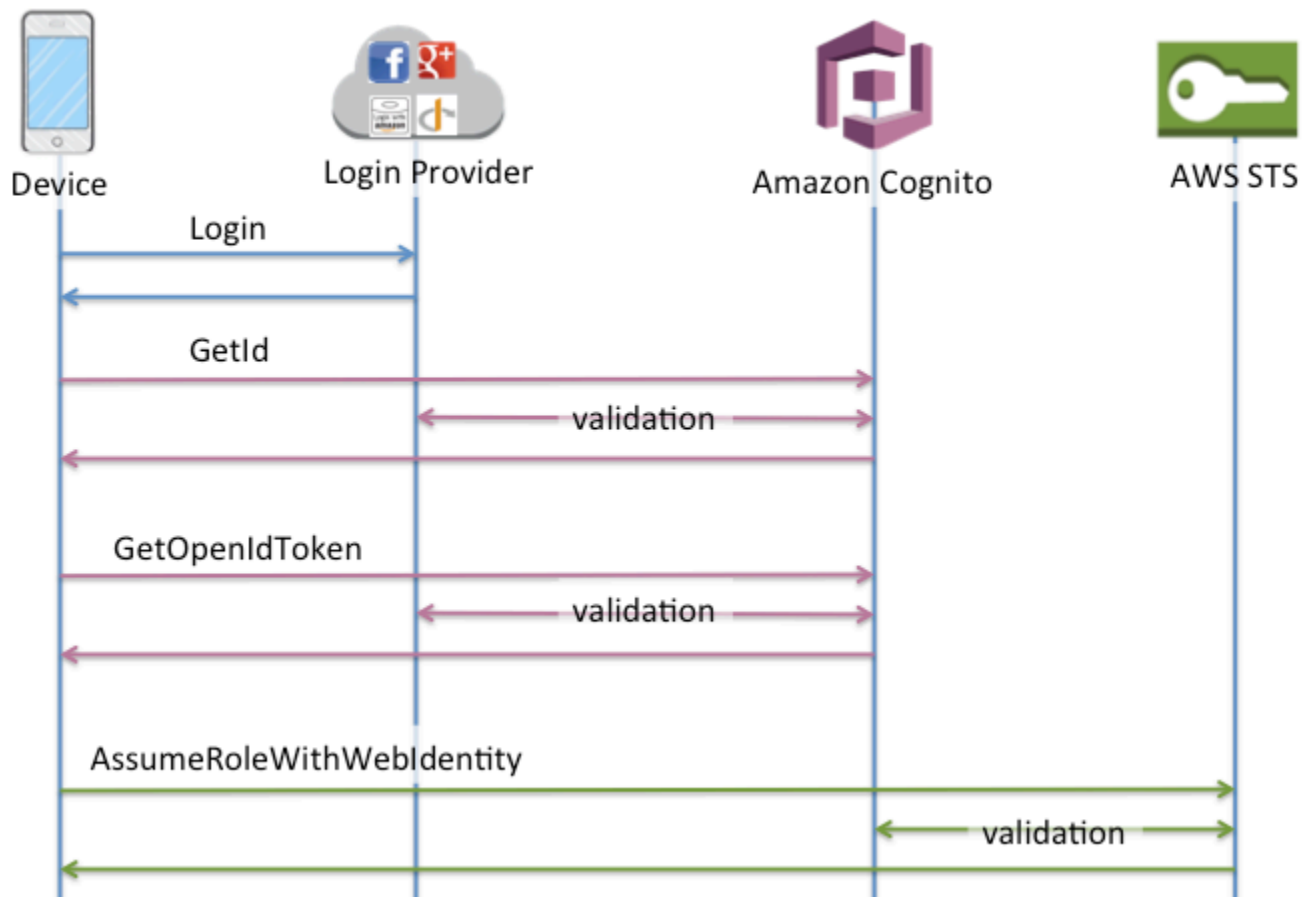
基本ワークフローでは、ユーザーに配布する認証情報をより細かく制御できます。拡張認証フローの GetCredentialsForIdentity リクエストは、アクセストークンの内容に基づいてロールをリクエストします。クラシックワークフローの AssumeRoleWithWebIdentity リクエストにより、十分な信頼ポリシーで設定した AWS Identity and Access Management ロールの認証情報をリクエストする機能がアプリに付与されます。カスタムロールセッション期間をリクエストすることもできます。

ロールマッピングがないユーザープールの基本認証フローを使用してサインインできます。このタイプのアイデンティティプールには、デフォルトの認証されているロールまたは認証されていないロールはなく、ロールベースのアクセス制御または属性ベースのアクセス制御も設定されていません。ロールマッピングを使用してアイデンティティプールで GetOpenIdToken を試行すると、次のエラーが表示されます。

Basic (classic) flow is not supported with RoleMappings, please use enhanced flow.

基本認証でのオペレーションの順序

1. GetId
2. GetOpenIdToken
3. AssumeRoleWithWebIdentity

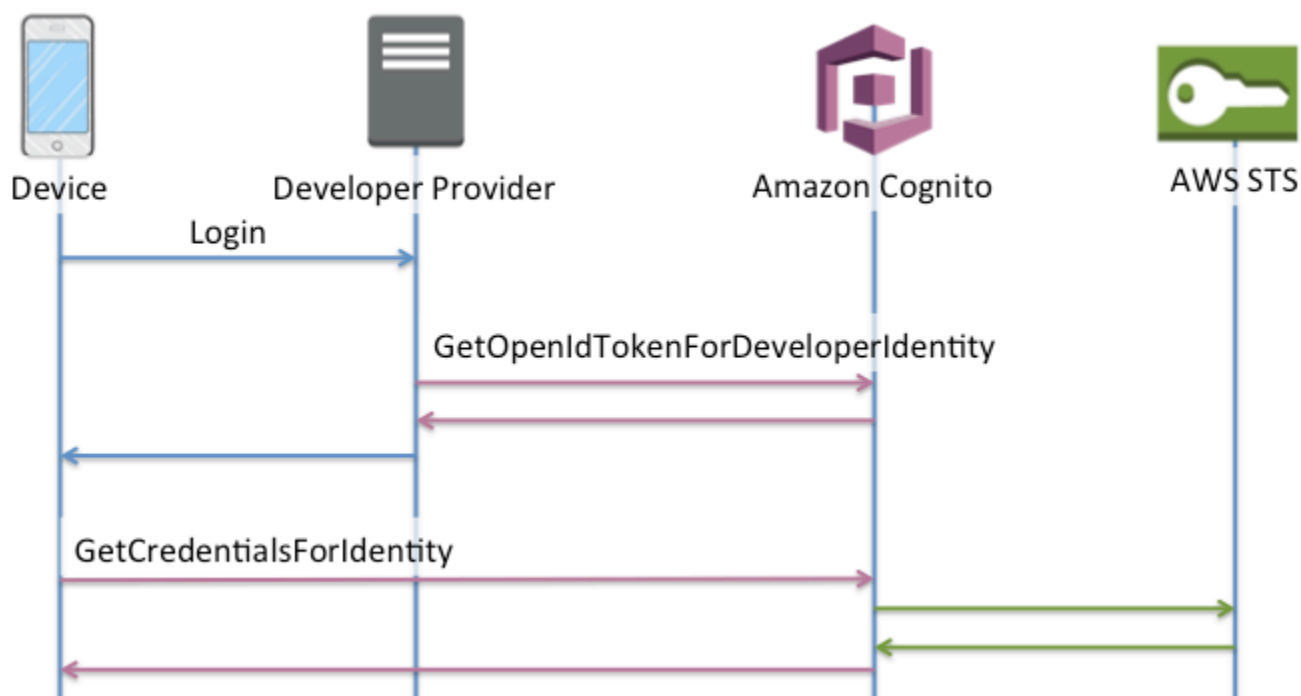


デベロッパーが認証した認証フロー

デベロッパーが認証した ID を使用する場合、クライアントは Amazon Cognito 外部のコードを含む別の認証フローを使用して、独自の認証システムでユーザーを検証します。アイデンティティプールの観点からは、ID のリクエストで提示したクレームは任意の識別子であり、認証はアプリケーションでエンコードした IAM 認証情報によって承認されます。

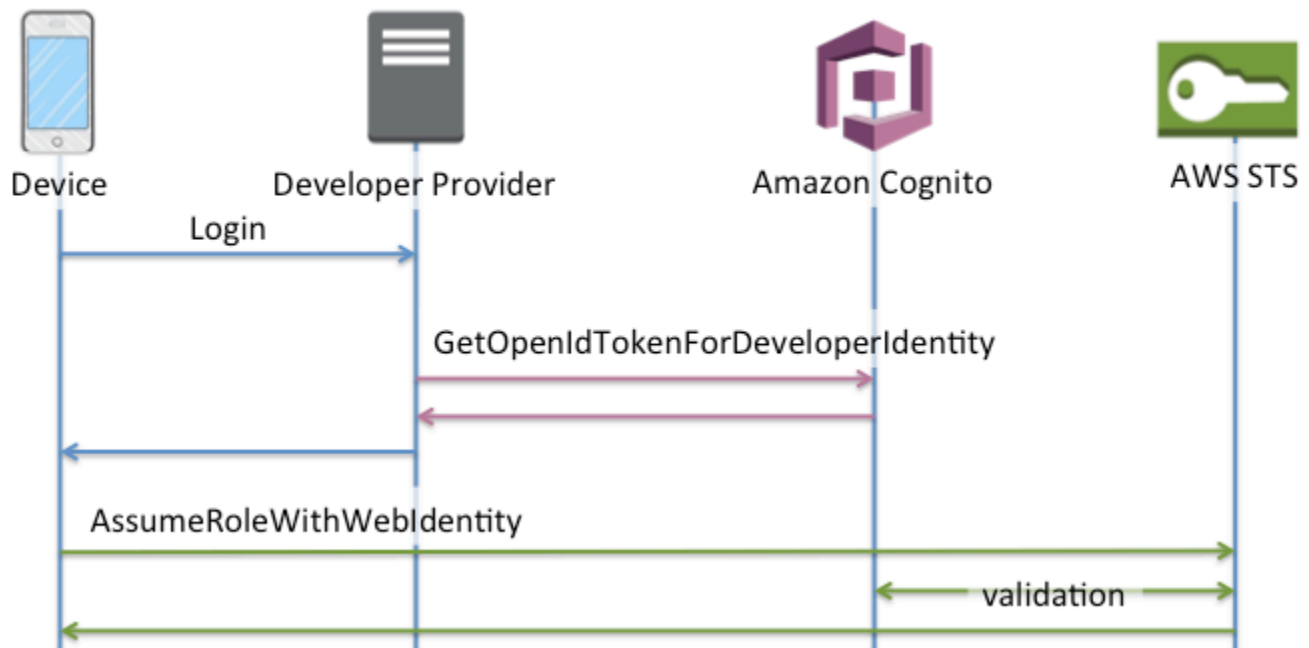
デベロッパープロバイダーによる拡張認証でのオペレーションの順序

1. デベロッパープロバイダー経由でのログイン (Amazon Cognito 外部のコード)
2. ユーザーログインの検証 (Amazon Cognito 外部のコード)
3. [GetOpenIdTokenForDeveloperIdentity](#)
4. [GetCredentialsForIdentity](#)



デベロッパープロバイダーによる基本認証でのオペレーションの順序

1. アイデンティティプールの外部にロジックを実装してサインインし、デベロッパープロバイダー識別子を生成します。
2. 保存されたサーバー側の AWS 認証情報を取得します。
3. 認可された AWS 認証情報で署名された [GetOpenIdTokenForDeveloperIdentity](#) API リクエストでデベロッパープロバイダー識別子を送信します。
4. [AssumeRoleWithWebIdentity](#) を使用してアプリケーション認証情報をリクエストします。



どの認証フローを実装する必要があるか？

拡張フローは、デベロッパーの手間が最も少ない、最も安全な選択肢です。

- 拡張フローにより、API リクエストの複雑さ、サイズ、レートが軽減されます。
- アプリケーションは、AWS STSに追加の API リクエストを行う必要はありません。
- アイデンティティプールは、ユーザーが受け取る必要がある IAM ロール認証情報についてユーザーを評価します。クライアントでロールを選択するためにロジックを埋め込む必要はありません。

⚠ Important

新しいアイデンティティプールを作成するときは、ベストプラクティスとして、基本 (クラシック) 認証をデフォルトでアクティブ化しないでください。基本認証を実装するには、まずウェブ ID の IAM ロールの信頼関係を評価します。次に、クライアントにロール選択のロジックを構築し、ユーザーによる変更からクライアントを保護します。

基本認証フローは、IAM ロール選択のロジックをアプリケーションに委任します。このフローでは、Amazon Cognito はユーザーの認証されたセッションまたは認証されていないセッションを検証し、認証情報と交換できるトークンを発行します AWS STS。ユーザーは、基本認証からのトークン

を、アイデンティティプールと `amr` を信頼する任意の IAM ロール、または認証されている状態/認証されていない状態と交換できます。

同様に、デベロッパー認証は ID プロバイダー認証の検証に関するショートカットであることを理解してください。Amazon Cognito は、リクエストの内容を追加で検証することなく、[GetOpenIdTokenForDeveloperIdentity](#) リクエストを認可する AWS 認証情報を信頼します。デベロッパー認証を認可するシークレットを、ユーザーによるアクセスから保護します。

認証フローの API オペレーションの概要

GetId

[GetId](#) API コールは、Amazon Cognito で新しい ID を確立するために必要な最初の呼び出しです。

認証されていないアクセス

Amazon Cognito には、アプリケーションで認証されていないゲストアクセスを許可できます。この機能が ID プールで有効になっている場合、ユーザーはいつでも GetId API 経由で新しいアイデンティティ ID をリクエストできます。Amazon Cognito に後続のコールを実行するため、アプリケーションにはこのアイデンティティ ID をキャッシュすることが期待されます。ブラウザの AWS Mobile SDKs と AWS SDK for JavaScript には、このキャッシュを処理する認証情報プロバイダーがあります。

認証されたアクセス

パブリックログインプロバイダー (Facebook、Google+、Login with Amazon、または「Apple でサインイン」など) のサポートでアプリケーションを設定すると、ユーザーは、これらのプロバイダーでユーザーを識別するトークン (OAuth または OpenID Connect) を提供できるようになります。GetId の呼び出しで使用されると、Amazon Cognito は新しい認証されたアイデンティティを作成するか、その特定のログインに既に関連付けられているアイデンティティを返します。Amazon Cognito は、プロバイダーでトークンを検証し、以下を確実にすることでこれを行います。

- トークンは有効で、設定されたプロバイダーからのものである。
- トークンの有効期限が切れていない。
- トークンがそのプロバイダーで作成されたアプリケーション識別子 (例えば、Facebook アプリ ID) と一致する。
- トークンがユーザー識別子と一致する。

GetCredentialsForIdentity

[GetCredentialsForIdentity](#) API は、アイデンティティ ID を確立した後で呼び出すことができます。このオペレーションは [GetOpenIdToken](#)、[AssumeRoleWithWebIdentity](#) の順に呼び出すのと機能的に同等です。

代わりに Amazon Cognito で [AssumeRoleWithWebIdentity](#) を呼び出す場合は、Amazon Cognito に関連付けられた IAM ロールが ID プールにある必要があります。これは、Amazon Cognito コンソールを使用して、または手動で [SetIdentityPoolRoles](#) オペレーションを使用して実行できます。

GetOpenIdToken

アイデンティティ ID を確立した後で [GetOpenIdToken](#) API リクエストを行います。最初のリクエスト後にアイデンティティ ID をキャッシュし、[GetOpenIdToken](#) を使用してその ID 以降の基本 (クラシック) セッションを開始します。

[GetOpenIdToken](#) API リクエストに対するレスポンスは、Amazon Cognito が生成するトークンです。このトークンは、[AssumeRoleWithWebIdentity](#) リクエストの `WebIdentityToken` パラメータとして送信できます。

OpenID トークンを送信する前に、アプリで検証してください。SDK の OIDC ライブラリや [aws-jwt-verify](#) のようなライブラリを使用して、Amazon Cognito がトークンを発行したことを確認できます。OpenID トークンの署名キー ID または `kid` は、Amazon Cognito ID [jwks_uri](#) [ドキュメント](#)† にリストされているもののいずれかです。これらのキーは変更される可能性があります。Amazon Cognito ID トークンを検証する関数は、`jwks_uri` ドキュメントからキーのリストを定期的に更新する必要があります。Amazon Cognito は、`jwks_uri` キャッシュコントロールレスポンスヘッダーで更新期間を設定しており、現在 `max-age` の 30 日間に設定されています。

認証されていないアクセス

認証されていない ID のトークンを取得するには、アイデンティティ ID そのもののみが必要です。認証された ID または無効にした ID の認証されていないトークンを取得することはできません。

認証されたアクセス

認証済みの ID がある場合、その ID に既に関連付けられたログイン用に、少なくとも 1 つのトークンを渡す必要があります。[GetOpenIdToken](#) の呼び出し中に渡されるすべてのトークンは、前に説明したのと同じ検証を渡す必要があります。いずれかのトークンが失敗すると、呼び出し全体が失敗します。[GetOpenIdToken](#) 呼び出しからの応答に、アイデンティティ ID

が含まれることもあります。これは、渡すアイデンティティ ID が、返される ID とは限らないためです。

ログインのリンク

既に任意の ID と関連付けられていないログインのトークンを送信すると、そのログインは関連付けられた ID に「リンクしている」と見なされます。パブリックプロバイダーごとに、1 つのログインのみをリンクできます。複数のログインをパブリックプロバイダーにリンクしようとすると、応答として `ResourceConflictException` エラーが発生します。ログインが単純に既存のアイデンティティにリンクされている場合、`GetOpenIdToken` から返されるアイデンティティ ID は、渡された ID と同じになります。

ID の結合

現在、特定の ID にリンクされていないが、別の ID にリンクされているログインに対してトークンを渡すと、2 つの ID が結合されます。いったん結合されると、1 つの ID は、すべての関連ログインの親/所有者になり、もう 1 つは無視されます。この場合、親/所有者のアイデンティティ ID が返されます。この値が異なる場合は、ローカルキャッシュを更新する必要があります。ブラウザの AWS Mobile SDKs または AWS SDK for JavaScript のプロバイダーが、このオペレーションを実行します。

`GetOpenIdTokenForDeveloperIdentity`

[GetOpenIdTokenForDeveloperIdentity](#) オペレーションは、デベロッパーが認証した ID を使用するとき、デバイスから [GetId](#) と [GetOpenIdToken](#) の使用を置き換えます。アプリケーションは AWS 認証情報を使用してこの API オペレーションへのリクエストに署名するため、Amazon Cognito はリクエストで指定されたユーザー識別子が有効であると信頼します。デベロッパー認証は、Amazon Cognito が実行するトークン検証を外部プロバイダーに置き換えます。

この API のペイロードには `logins` マップが含まれています。このマップには、デベロッパープロバイダーのキーとシステムでのユーザーの識別子としての値が含まれている必要があります。ユーザー識別子がまだ既存のアイデンティティに既にリンクされていない場合は、Amazon Cognito が新しいアイデンティティを作成して、新しいアイデンティティ ID と、そのアイデンティティの OpenID Connect を返します。ユーザー識別子が既にリンクされている場合は、Amazon Cognito が既存のアイデンティティ ID と OpenID Connect トークンを返します。最初のリクエスト後にデベロッパーアイデンティティ ID をキャッシュし、`GetOpenIdTokenForDeveloperIdentity` を使用してその ID 以降の基本 (クラシック) セッションを開始します。

GetOpenIdTokenForDeveloperIdentity API リクエストに対するレスポンスは、Amazon Cognito が生成するトークンです。このトークンは、AssumeRoleWithWebIdentity リクエストの WebIdentityToken パラメータとして送信できます。

OpenID Connect トークンを送信する前に、アプリで検証してください。SDK の OIDC ライブラリや [aws-jwt-verify](#) のようなライブラリを使用して、Amazon Cognito がトークンを発行したことを確認できます。OpenID Connect トークンの署名キー ID または kid は、Amazon Cognito ID [jwks_uri ドキュメント](#) にリストされているもののいずれかです。これらのキーは変更される可能性があります。Amazon Cognito ID トークンを検証する関数は、jwks_uri ドキュメントからキーのリストを定期的に更新する必要があります。Amazon Cognito は、jwks_uri cache-control レスポンスヘッダーで更新期間を設定しており、現在 max-age の 30 日間に設定されています。

ログインのリンク

外部プロバイダーと同様に、既に ID に関連付けられていない追加のログインを指定すると、それらのログインがその ID に暗黙的にリンクされます。外部プロバイダーのログインを ID にリンクする場合、ユーザーはそのプロバイダーで外部プロバイダーの認証フローを使用できます。ただし、GetId または GetOpenIdToken を呼び出すときに、ログインマップで開発者プロバイダ名を使用することはできません。

ID の結合

Amazon Cognito は、デベロッパーが認証したアイデンティティを使用し、[MergeDeveloperIdentities](#) API を通じて暗黙的なマージと明示的なマージの両方をサポートします。明示的なマージにより、システムでユーザー識別子を持つ 2 つのアイデンティティを、1 つのアイデンティティとしてマークできます。ソースと宛先のユーザー識別子を指定すると、Amazon Cognito がそれらをマージします。次回にいずれかのユーザー識別子に対して OpenID Connect トークンをリクエストすると、同じアイデンティティ ID が返されます。

AssumeRoleWithWebIdentity

OpenID Connect トークンを取得したら、AWS Security Token Service () への [AssumeRoleWithWebIdentity](#) API リクエストを通じて、これを一時的な AWS 認証情報と交換できます AWS STS。

作成できる ID の数には制限がないため、ユーザーに付与するアクセス権限を理解することが重要です。アプリケーション用の異なる IAM ロールを設定します (1 つは認証されていないユーザー用で、もう 1 つは認証されているユーザー用)。Amazon Cognito コンソールは、最初にアイ

デンティティプールをセットアップするときにデフォルトロールを作成できます。これらのロールには、アクセス許可が有効に付与されていません。ニーズに合わせて変更します。

[ロールの信頼とアクセス権限](#) の詳細を確認してください。

† デフォルトの Amazon Cognito ID の [jwks_uri](#) ドキュメントには、ほとんどの AWS リージョンで ID プールのトークンに署名するキーに関する情報が含まれています。以下のリージョンには異なる jwks_uri ドキュメントがあります。

Amazon Cognito Identity JSON web key URIs in other AWS リージョン

AWS リージョン	jwks_uri ドキュメントへのパス
AWS GovCloud (US-West)	<code>https://cognito-identity.us-gov-west-1.amazonaws.com/.well-known/jwks_uri</code>
China (Beijing)	<code>https://cognito-identity.cn-north-1.amazonaws.com.cn/.well-known/jwks_uri</code>
Opt-in Regions like Europe (Milan) and Africa (Cape Town)	<code>https://cognito-identity. <i>Region</i>.amazonaws.com/.well-known/jwks_uri</code>

また、発行者から jwks_uri を推定することも、Amazon Cognito から OpenID トークンで受け取った iss を推定することもできます。OIDC 標準の検出エンドポイント <issuer>/.well-known/openid-configuration には、トークンの jwks_uri へのパスがリストされます。

IAM ロール

ID プールの作成中に、ユーザーが引き受ける IAM ロールを更新するよう求められます。IAM ロールは次のように動作します。ユーザーがアプリにログインすると、Amazon Cognito がそのユーザー用に一時的な AWS 認証情報を生成します。これらの一時的な認証情報は、特定の IAM ロールに関連付けられます。IAM ロールでは、AWS リソースにアクセスするための許可一식을定義できます。

認証されたユーザーと認証されていないユーザー用に、デフォルトの IAM ロールを指定できます。また、ユーザーの ID トークンのクレームに基づいて、各ユーザーのロールを選択するルールを定義できます。詳細については、「[ロールベースアクセスコントロールの使用](#)」を参照してください。

Amazon Cognito コンソールは、Amazon Mobile Analytics および Amazon Cognito Sync へのアクセス権を提供する IAM ロールをデフォルトで作成します。または、既存の IAM ロールの使用を選択できます。

IAM ロールを変更して、他のサービスへのアクセスを許可または制限します。これを行うためには、[IAM コンソールにログインします](#)。次に、[Roles] (ロール) を選択し、ロールを選択します。選択されたロールにアタッチされたポリシーが、[Permissions] (アクセス許可) タブに表示されます。対応する [Manage Policy] (ポリシーの管理) リンクを選択してアクセスポリシーをカスタマイズできます。ポリシーの使用と定義の詳細については、「[IAM ポリシーの概要](#)」を参照してください。

Note

ベストプラクティスとして、最小権限の付与の原則に準拠したポリシーを定義します。つまり、ポリシーには、ユーザーがタスクを実行するために必要なアクセス権限のみを含めます。詳細については、「IAM ユーザーガイド」の「[Grant Least Privilege](#)」(最小権限を付与する) を参照してください。

認証されていない ID は、アプリにログインしていないユーザーが引き受けることに注意してください。通常、認証されていない ID に割り当てるアクセス権限は、認証された ID に割り当てるアクセス権限よりも制限が厳しいものである必要があります。

トピック

- [信頼ポリシーのセットアップ](#)
- [アクセスポリシー](#)
- [ロールの信頼とアクセス権限](#)

信頼ポリシーのセットアップ

Amazon Cognito は IAM ロールを使用して、アプリケーションのユーザー用の一時的な認証情報を生成します。アクセス権限へのアクセスは、ロールの信頼関係によって制御されます。詳細情報 [ロールの信頼とアクセス権限](#)。Amazon Cognito は、AWS STS と ID プール IdPs。

に提示されるトークン AWS STS は、ユーザープール、ソーシャルまたは OIDC プロバイダートークン、または SAML アサーションを独自のトークンに変換する ID プールによって生成されます。ア

アイデンティティプールトークンには、アイデンティティプール ID として aud クレームが含まれています。

IAM ロール信頼ポリシーの Principal が `cognito-identity.amazonaws.com` のようなアイデンティティプールサービスプリンシパルである場合、アイデンティティプールがロールを引き受けることを許可するようにロール信頼ポリシーを作成または変更することはできません。アイデンティティプールのプリンシパルの場合、Action 要素には Condition が必要であり、これによって、(`cognito-identity.amazonaws.com:aud` などの条件キーで指定したとおり) アイデンティティプールのみが `AssumeRoleWithWebIdentity` を実行するよう要求します。他の条件キーも使用できますが、aud は必須です。このタイプの条件なしでロール信頼ポリシーを保存しようとすると、IAM はエラーを返します。

OIDC (ウェブ ID) フェデレーションキーの詳細については、[「OIDC AWS フェデレーションで利用できるキー」](#)を参照してください。

Amazon Cognito で使用できる OIDC フェデレーション条件キーは、以下のとおりです。

`cognito-identity.amazonaws.com:aud`

ロールを 1 つ以上のアイデンティティプールからのオペレーションに制限します。Amazon Cognito は、アイデンティティプールトークンの aud クレーム内のソース ID プールを示します。

`cognito-identity.amazonaws.com:amr`

ロールを `authenticated` または `unauthenticated` (ゲスト) ユーザーに制限します。Amazon Cognito は、アイデンティティプールトークンの amr クレームの認証状態を示します。

`cognito-identity.amazonaws.com:sub`

[UUID](#) によってロールを 1 人以上のユーザーに制限します。この UUID は、アイデンティティプール内のユーザーのアイデンティティ ID です。この値は、ユーザーの元の ID プロバイダーの sub 値ではありません。Amazon Cognito は、アイデンティティプールトークンの sub クレームにこの UUID を示します。

次のロール信頼ポリシーの例では、フェデレーテッドサービスプリンシパルが AWS STS API `cognito-identity.amazonaws.com` を呼び出すことを許可します `AssumeRoleWithWebIdentity`。リクエストが成功するのは、API リクエストのアイデンティティプールトークンに以下のクレームが含まれている場合のみです。

1. アイデンティティプール ID `aud` の `us-west-2:abcdefg-1234-5678-910a-0e8443553f95` クレーム。
2. ユーザーがログインしていて、ゲストユーザーではない場合に追加される `authenticated` の `amr` クレーム。

JSON

```
{
  "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"
        }
      }
    }
  ]
}
```

基本 (クラシック) 認証での IAM ロールの信頼ポリシー

概要

アイデンティティプールは、ターゲットのロール信頼ポリシーに `aud` 条件が含まれている場合にのみ、[基本認証フロー](#)でユーザーに代わってロールを引き受けることができます。

基本認証には、拡張認証と同じように、安全でないロール信頼ポリシーに関する制限があります。つまり、サポートされているアイデンティティプールを `aud` 条件で制限できないロール信頼ポリシーを保存できません。この制限は、サービスのリリース時には適用されていませんでした。この要件

の適用前は、追加のセキュリティ条件がないロール信頼ポリシーを作成できました。この要件の適用後、は、条件で保護されていないロールを引き受けるウェブ ID AWS STS を許可しますが、これらの条件を導入しない限り、それらのロールを変更することはできません。

拡張フロー認証では、IAM ロールがアイデンティティプールと同じ AWS アカウント に存在する必要があります。一方、基本認証では、アプリケーションが AssumeRoleWithWebIdentity リクエストを作成し、別のアカウントのロールを引き受けることをリクエストできます。ただし、送信先ロールに aud 条件を適用しないレガシー信頼ポリシーがある場合、[クロスアカウント](#)のロール引き受けリクエストは失敗します。

ID プールが ID に対して発行するトークンには、ID プール AWS アカウント の発信元に関する情報が含まれています。[AssumeRoleWithWebIdentity](#) API リクエストで ID プールトークンを提示すると、は、発信元の ID プールが IAM ロール AWS アカウント と同じ にある AWS STS かどうかを確認します。がリクエストがクロスアカウントである AWS STS と判断した場合、ロール信頼ポリシーに aud 条件があるかどうかをチェックします。そのような条件がロール信頼ポリシーに存在しない場合、ロール引き受け呼び出しは失敗します。リクエストがクロスアカウントでない場合は、この制限を適用 AWS STS しません。ベストプラクティスとして、このタイプの条件をアイデンティティプールロールの信頼ポリシーに常に適用します。

次に示すのは、複数のアイデンティティプールを使用した基本認証において、IAM ロールの最小要件を満たす信頼ポリシーの例です。ベストプラクティスとして、"cognito-identity.amazonaws.com:amr": "authenticated" 条件でも認証済み ID のみを許可するようにします。

JSON

```
{
  "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",
            "us-west-2:ijklmo-5678-9101-112b-0e4221776g96",
```



```
"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"
  ]
}
```

特定のプロバイダーへのアクセスの制限

特定のプロバイダー (おそらくはユーザー独自のログインプロバイダー) でログインしたユーザーに制限されたポリシーを作成するには、`cognito-identity.amazonaws.com:amr` の値を確認します。

```
"ForAnyValue:StringLike": {
  "cognito-identity.amazonaws.com:amr": "login.myprovider.myapp"
}
```

たとえば、Facebook のみを信頼するアプリに、次の `amr` 句があるとします。

```
"ForAnyValue:StringLike": {
  "cognito-identity.amazonaws.com:amr": "graph.facebook.com"
}
```

アクセスポリシー

ロールにアタッチするアクセス許可は、そのロールを引き受けるすべてのユーザーに適用されます。ユーザーのアクセスをパーティション化するには、ポリシー条件と変数を使用します。詳細については、「[IAM ポリシーの要素: 変数とタグ](#)」を参照してください。sub 条件を使用して、アクションをアクセスポリシーの Amazon Cognito アイデンティティ ID に制限できます。このオプションは注意して使用してください。特に、一貫したユーザー ID を持たない認証されていない ID には注意が必要です。Amazon Cognito とのウェブフェデレーションの IAM ポリシー変数の詳細については、AWS Identity and Access Management 「ユーザーガイド」の「[IAM および AWS STS 条件コンテキストキー](#)」を参照してください。

セキュリティ保護を向上させるために、Amazon Cognito は `GetCredentialsForIdentity` を使用して、[拡張フロー](#)で認証されていないユーザーを割り当てる認証情報にスコープダウンポリシーを適用します。スコープダウンポリシーは、認証されていないロールに適用する IAM ポリシーに [インラインセッションポリシー](#) と [AWS マネージドセッションポリシー](#) を追加します。ロールの IAM ポ

リシーとセッションポリシーの両方でアクセスを許可する必要があるため、スコープダウンポリシーでは、次のリストに示すサービス以外のサービスへのユーザーアクセスが制限されます。

Note

基本 (クラシック) フローでは、独自の [AssumeRoleWithWebIdentity](#) API リクエストを作成し、そのリクエストにこれらの制限を適用できます。セキュリティのベストプラクティスとして、このスコープダウンポリシーを超えるアクセス許可を認証されていないユーザーに割り当てないでください。

また、Amazon Cognito は、認証されたユーザーと認証されていないユーザーが Amazon Cognito ID プールと Amazon Cognito Sync に API リクエストを実行できないようにします。他の は、ウェブ ID からのサービスアクセスを制限する AWS のサービス 場合があります。

拡張フローでリクエストが成功すると、Amazon Cognito はバックグラウンドで AssumeRoleWithWebIdentity API リクエストを行います。このリクエストのパラメータとしては、Amazon Cognito に次のものが含まれます。

1. ユーザーのアイデンティティ ID。
2. ユーザーが引き受ける IAM ロールの ARN。
3. インラインセッションポリシーを追加する policy パラメータ。
4. Amazon CloudWatch で追加のアクセス許可を付与する AWS マネージドポリシーを値として持つ PolicyArns.member.N パラメータ。

認証されていないユーザーがアクセスできるサービス

拡張フローを使用すると、Amazon Cognito がユーザーセッションに適用するスコープダウンポリシーにより、次の表に記載されている以外のサービスを使用できなくなります。一部のサービスでは、特定のアクションのみが許可されます。

Category	サービス
分析	Amazon Data Firehose
	Amazon Managed Service for Apache Flink
アプリケーション統合	Amazon Simple Queue Service

Category	サービス
AR とバーチャルリアリティ	Amazon Sumerian ¹
ビジネスアプリケーション	Amazon Mobile Analytics Amazon Simple Email Service
コンピューティング	AWS Lambda
暗号化と PKI	AWS Key Management Service 1
データベース	Amazon DynamoDB Amazon SimpleDB
フロントエンドウェブおよびモバイル	AWS AppSync Amazon Location Service Amazon Simple Notification Service Amazon Pinpoint Amazon Location Service
ゲーム開発	Amazon GameLift Servers
IoT	AWS IoT

Category	サービス
機械学習	Amazon CodeWhisperer
	Amazon Comprehend
	Amazon Lex
	Amazon Machine Learning
	Amazon Personalize
	Amazon Polly
	Amazon Rekognition
	Amazon SageMaker AI ¹
	Amazon Textract ¹
	Amazon Transcribe
	Amazon Translate
管理とガバナンス	Amazon CloudWatch
	Amazon CloudWatch Logs
ネットワーキングとコンテンツ配信	Amazon API Gateway
セキュリティ、アイデンティティ、コンプライアンス	Amazon Cognito ユーザープール
Storage	Amazon Simple Storage Service

1 次の表の AWS のサービス の場合、インラインポリシーはアクションのサブセットを付与します。表には、それぞれで実行可能なアクションが表示されています。

AWS のサービス	認証されていない拡張フローユーザーの最大アクセス許可
AWS Key Management Service	Encrypt
	Decrypt
	ReEncryptTo
	ReEncryptFrom
	GenerateDataKey
	GenerateDataKeyPair
	GenerateDataKeyPair
	GenerateDataKeyPairWithoutPlaintext
	GenerateDataKeyWithoutPlaintext
Amazon SageMaker AI	InvokeEndpoint
Amazon Textract	DetectDocumentText
	AnalyzeDocument
Amazon Sumerian	View*
Amazon Location Service	SearchPlaceIndex*
	GetPlace
	CalculateRoute*
	*Geofence
	*Geofences
	DevicePosition

このリスト AWS のサービス 以外の へのアクセスを許可するには、ID プールで基本 (クラシック) 認証フローをアクティブ化します。認証されていないユーザーの IAM ロールに割り当てられたポリシーで許可されている `NotAuthorizedException` エラー AWS のサービス が からユーザーに表示される場合は、そのサービスをユースケースから削除できるかどうかを評価します。できない場合は、基本フローに切り替えてください。

ゲストユーザー用のインラインセッションポリシー

Amazon Cognito はまず、IAM 認証情報のリクエストにインラインポリシーを適用します。インラインセッションポリシーでは、ユーザーの有効な権限に、以下のリストにある AWS のサービス 外部へのアクセスが含まれないように制限しています。また、ユーザーの IAM ロールに適用するポリシー AWS のサービス で、これらにアクセス許可を付与する必要があります。ロールを引き受けたセッションにおけるユーザーの有効なアクセス許可は、そのユーザーのロールに割り当てられたポリシーとセッションポリシーが交差したものです。詳細については、AWS Identity and Access Management ユーザーガイドの「[セッションポリシー](#)」を参照してください。

Amazon Cognito は、デフォルトで有効になっている AWS リージョン ユーザーのセッションに次のインラインポリシーを追加します。インラインポリシーおよびその他のセッションポリシーによる正味の効果の概要については、「[認証されていないユーザーがアクセスできるサービス](#)」を参照してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:*",
        "logs:*",
        "dynamodb:*",
        "kinesis:*",
        "mobileanalytics:*",
        "s3:*",
        "ses:*",
        "sns:*",
        "sqs:*",
        "lambda:*",
        "machinelearning:*
```

```

        "execute-api:*",
        "iot:*",
        "gamelift:*",
        "cognito-identity:*",
        "cognito-idp:*",
        "lex:*",
        "polly:*",
        "comprehend:*",
        "translate:*",
        "transcribe:*",
        "rekognition:*",
        "mobiletargeting:*",
        "firehose:*",
        "appsync:*",
        "personalize:*",
        "sagemaker:InvokeEndpoint",
        "cognito-sync:*",
        "codewhisperer:*",
        "textract:DetectDocumentText",
        "textract:AnalyzeDocument",
        "sdb:*"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

他のすべてのリージョンの場合、インラインスコープダウンポリシーには、次の Action ステートメントを除いて、デフォルトのリージョンにリストされているものがすべて含まれます。

```

        "cognito-sync:*",
        "sumerian:View*",
        "codewhisperer:*",
        "textract:DetectDocumentText",
        "textract:AnalyzeDocument",
        "sdb:*"

```

ゲストの AWS マネージドセッションポリシー

また、Amazon Cognito は、認証されていないゲストの拡張フローセッションにセッションポリシーとして AWS 管理ポリシーを適用します。このポリシーは、ポリシー `AmazonCognitoUnAuthedIdentitiesSessionPolicy` を使用して、認証されていないユーザーのアクセス許可の範囲を制限します。

また、認証されていない IAM ロールに適用するポリシーでこのアクセス許可を付与する必要があります。ロールを引き受けたセッションにおけるユーザーの有効なアクセス許可は、そのユーザーのロールに割り当てられた IAM ポリシーとセッションポリシーが交差したものです。詳細については、AWS Identity and Access Management ユーザーガイドの「[セッションポリシー](#)」を参照してください。

この AWS 管理ポリシーおよびその他のセッションポリシーの正味効果の概要については、「」を参照してください [認証されていないユーザーがアクセスできるサービス](#)。

`AmazonCognitoUnAuthedIdentitiesSessionPolicy` 管理ポリシーには以下のアクセス許可が記載されています。

JSON

```
{
  "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": "*"
}]
}
```

アクセスポリシーの例

このセクションでは、特定のオペレーションの実行に必要な最小限のアクセス許可を付与する Amazon Cognito アクセスポリシーの例を示します。可能な場合は、ポリシー変数を使用して特定のアイデンティティ ID のアクセス権限をさらに制限できます。たとえば、`${cognito-identity.amazonaws.com:sub}` を使用します。詳細については、AWS モバイルブログで

「[Understanding Amazon Cognito Authentication Part 3: Roles and Policies](#)」を参照してください。

Note

セキュリティのベストプラクティスとして、ポリシーには、ユーザーがタスクを実行するために必要なアクセス権限のみを含めてください。つまり、可能であれば必ずオブジェクトの個々の ID ごとにアクセスを設定するようにしてください。

ID に Amazon S3 内の単一のオブジェクトへの読み取りアクセスを許可する

次のアクセスポリシーは、特定の S3 バケットから 1 つのオブジェクトを取得するための読み取りアクセス権限を ID に付与します。

JSON

```
{
  "Version": "2012-10-17",
```

```

    "Statement": [
      {
        "Action": [
          "s3:GetObject"
        ],
        "Effect": "Allow",
        "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/assets/my_picture.jpg"]
      }
    ]
  }
}

```

ID に Amazon S3 の ID 固有のパスへの読み取りおよび書き込みアクセスを許可する

次のアクセスポリシーは、プレフィックスを `${cognito-identity.amazonaws.com:sub}` 変数にマッピングすることで、S3 バケット内の特定のプレフィックス「folder」にアクセスする読み取りおよび書き込みアクセス権限を付与します。

このポリシーでは、`${cognito-identity.amazonaws.com:sub}` を使用して挿入された `us-east-1:12345678-1234-1234-1234-123456790ab` などの ID は、オブジェクトを取得し、`arn:aws:s3:::amzn-s3-demo-bucket/us-east-1:12345678-1234-1234-1234-123456790ab` に配置およびリストできます。ただし、この ID には `arn:aws:s3:::amzn-s3-demo-bucket` の他のオブジェクトへのアクセスは許可されません。

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${cognito-identity.amazonaws.com:sub}/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ]
    }
  ]
}

```

```
    ],
    "Effect": "Allow",
    "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket/${cognito-identity.amazonaws.com:sub}/*"]
  }
]
```

同様のアクセスモデルは、[Amazon S3 Access Grants](#) で実現されます。

ID に Amazon DynamoDB への詳細に設定されたアクセスを割り当てる

次のアクセスポリシーは、Amazon Cognito 環境変数を使用して、Amazon DynamoDB リソースへの詳細に設定されたアクセスコントロールを行います。これらの変数は、アイデンティティ ID によって DynamoDB のアイテムへのアクセス権限を付与します。詳細については、Amazon DynamoDB デベロッパーガイドの「[詳細に設定されたアクセスコントロールのための IAM ポリシー条件の使用](#)」を参照してください。

JSON

```
{
  "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}"]
        }
      }
    }
  ]
}
```

```
}  
}  
]  
}
```

Lambda 関数を呼び出すためのアクセス権限を ID に与えます。

次のアクセスポリシーは、Lambda 関数を呼び出すアクセス権限を ID に付与します。

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "lambda:InvokeFunction",  
      "Resource": [  
        "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"  
      ]  
    }  
  ]  
}
```

Amazon Kinesis Data Stream へのレコードを発行するアクセス権限を ID に付与する

次のアクセスポリシーでは、ID が任意の Kinesis Data Streams で PutRecord オペレーションを使用することを許可します。これは、アカウント内のすべてのストリームにデータレコードを追加する必要があるユーザーに適用できます。詳細については、Amazon Kinesis Data Streams デベロッパーガイドの「[IAM を使用して Amazon Kinesis Streams リソースへのアクセスを制御する](#)」を参照してください。

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",
```

```
        "Action": "kinesis:PutRecord",
        "Resource": [
            "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
        ]
    }
]
```

ID に Amazon Cognito Sync ストアのデータへのアクセス権を付与する

以下のアクセスポリシーは、アイデンティティに Amazon Cognito Sync ストア内のデータのみに対するアクセス権限を付与します。

JSON

```
{
  "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}/*"]
  }]
}
```

ロールの信頼とアクセス権限

これらのロールが異なる点は、それらの信頼関係です。未認証ロールの信頼ポリシーの例を次に示します。

JSON

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

このポリシーは、cognito-identity.amazonaws.com (OpenID Connect トークンの発行者) のフェデレーテッドユーザーに、このロールを引き受ける許可を与えます。さらに、ポリシーでは、ID プールに一致させるためにトークンの aud が制限されます (この場合は ID プール ID)。最後に、このポリシーは、Amazon Cognito GetOpenIdToken API オペレーションによって発行されたトークンの複数の値からなる amr クレームの配列メンバーの 1 つが unauthenticated 値を含みます

Amazon Cognito がトークンを作成すると、トークンの amr を unauthenticated または authenticated のいずれかに設定します。amr が authenticated の場合、トークンには、認証中に使用されるすべてのプロバイダーが含まれます。これにより、amr 条件を次のように変更するだけで、Facebook 経由でログインしたユーザーのみを信頼するロールを作成できます。

```
"ForAnyValue:StringLike": {
  "cognito-identity.amazonaws.com:amr": "graph.facebook.com"
}
```

ロールで信頼関係を変更するときや、ID プール間でロールの使用を試みる場合は注意が必要です。ID プールを正しく信頼するようにロールが設定されていない場合は、STS の結果から次のような例外が表示されます。

```
AccessDenied -- Not authorized to perform sts:AssumeRoleWithWebIdentity
```

このメッセージが表示された場合は、ID プールと認証タイプに適切なロールがあることを確認します。

Amazon Cognito アイデンティティプールに対するセキュリティのベストプラクティス

Amazon Cognito ID プールは、アプリケーションの一時的な AWS 認証情報を提供します。AWS アカウントには、多くの場合、アプリケーションユーザーが必要とするリソースとプライベートバックエンドリソースの両方が含まれます。AWS 認証情報を構成する IAM ロールとポリシーは、これらのリソースへのアクセスを許可できます。

アイデンティティプール設定の主なベストプラクティスは、過剰な権限や意図しない権限なしでアプリケーションがジョブを完了できるようにすることです。セキュリティの設定ミスを防ぐために、本番環境にリリースする各アプリケーションの起動前に、これらの推奨事項を確認してください。

トピック

- [IAM 設定のベストプラクティス](#)
- [アイデンティティプール設定のベストプラクティス](#)

IAM 設定のベストプラクティス

ゲストまたは認証されたユーザーが、アイデンティティプールの認証情報を必要とするセッションをアプリケーションで開始すると、アプリケーションは IAM ロールの一時的な AWS 認証情報を取得します。認証情報は、デフォルトのロールのもの、アイデンティティプール設定のルールによって選択されたロールのもの、またはアプリケーションによって選択されたカスタムロールのものである場合があります。各ロールに割り当てられたアクセス許可により、ユーザーは AWS リソースにアクセスできます。

一般的な IAM のベストプラクティスの詳細については、AWS Identity and Access Management「ユーザーガイド」の「[IAM のベストプラクティス](#)」を参照してください。

IAM ロールで信頼ポリシー条件を使用する

IAM では、アイデンティティプールのロールに少なくとも 1 つの信頼ポリシー条件が必要です。この条件は、例えば、ロールの範囲を認証されたユーザーのみに設定することができます。AWS STS または、クロスアカウントの基本認証リクエストに

cognito-identity.amazonaws.com:audと の 2 つの特定の条件が必要ですcognito-identity.amazonaws.com:amr。ベストプラクティスとしては、アイデンティティプールサービスプリンシパル cognito-identity.amazonaws.com を信頼するすべての IAM ロールにこの両方の条件を適用します。

- cognito-identity.amazonaws.com:aud: アイデンティティプールトークンの aud クレームは、信頼されたアイデンティティプール ID と一致する必要があります。
- cognito-identity.amazonaws.com:amr: アイデンティティプールトークンの amr クレームは、認証されているもの、または認証されていないものである必要があります。この条件では、認証されていないゲストのみに、または認証されているユーザーのみにロールへのアクセスを予約できます。この条件の値をさらに絞り込んで、graph.facebook.com などの特定のプロバイダーのユーザーにロールを制限できます。

次のロール信頼ポリシーの例では、以下の条件下でロールへのアクセスを許可します。

JSON

```
{
  "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:a1b2c3d4-5678-90abcdef-EXAMPLE11111"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "authenticated"
        }
      }
    }
  ]
}
```

アイデンティティプールに関連する要素

- "Federated": "cognito-identity.amazonaws.com": ユーザーは、アイデンティティプールに帰属している必要があります。
- "cognito-identity.amazonaws.com:aud": "us-east-1:a1b2c3d4-5678-90ab-cdef-example11111": ユーザーは、特定のアイデンティティプール us-east-1:a1b2c3d4-5678-90ab-cdef-example11111 に帰属している必要があります。
- "cognito-identity.amazonaws.com:amr": "authenticated": ユーザーは、認証されている必要があります。ゲストユーザーはロールを引き受けることはできません。

最小特権アクセス許可を適用する

認証アクセスまたはゲストアクセスについて IAM ポリシーでアクセス許可を設定する場合、特定のタスクの実行に必要な特定のアクセス許可、または最小特権アクセス許可のみを付与します。次の IAM ポリシーの例では、これをロールに適用すると、Amazon S3 バケット内の 1 つの画像ファイルへの読み取り専用アクセスを許可します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/assets/my_picture.jpg"]
    }
  ]
}
```

アイデンティティプール設定のベストプラクティス

ID プールには、AWS 認証情報を生成するための柔軟なオプションがあります。アプリケーションが最も安全な方法で動作できる場合は、設計ショートカットを使用しないでください。

ゲストアクセスの効果を理解する

認証されていないゲストアクセスにより、ユーザーはサインインする前に AWS アカウント からデータを取得できます。アイデンティティプール ID を知っている者は誰でも、認証されていない認証情報をリクエストできます。アイデンティティプール ID は機密情報ではありません。ゲストアクセスをアクティブ化すると、認証されていないセッションに付与する AWS アクセス許可は誰でも利用できます。

ベストプラクティスとしては、ゲストアクセスを非アクティブ化のままにしておき、ユーザーが認証された後にのみ必要なリソースを取得します。アプリケーションがサインイン前にリソースにアクセスする必要がある場合は、次の予防措置を講じます。

- [認証されていないロールに課される自動制限](#)について十分に理解してください。
- アプリケーションの特定のニーズに合わせて、認証されていない IAM ロールのアクセス許可の監視と調整を行います。
- エンコードされたバージョンを使用するものです。特定のリソースへのアクセス許可を付与します。
- デフォルトの認証されていない IAM ロールの信頼ポリシーを保護します。
- インターネット上のすべてのユーザーに IAM ロールのアクセス許可を付与してもよいと確信できる場合にのみ、ゲストアクセスを有効にします。

デフォルトで拡張認証を使用する

基本 (クラシック) 認証では、Amazon Cognito は IAM ロールの選択をアプリケーションに委任します。対照的に、拡張フローは、アイデンティティプール内の一元化されたロジックを使用して IAM ロールを決定します。また、IAM アクセス許可の上限を設定する [スコープダウンポリシー](#) を使用して、認証されていない ID のセキュリティを強化します。拡張フローは、デベロッパーの手間が最も少ない、最も安全な選択肢です。これらのオプションの詳細については、「[ID プールの認証フロー](#)」を参照してください。

基本的なフローでは、認証情報の AWS STS API リクエストのロール選択とアセンブリに入るクライアント側のロジックを公開できます。拡張フローでは、アイデンティティプールの自動化の背後にあるロジックと assume-role リクエストの両方が非表示になります。

基本認証を設定するときは、IAM ロールとそのアクセス許可に [IAM ベストプラクティス](#) を適用します。

デベロッパープロバイダーを安全に使用する

デベロッパー認証 ID は、サーバー側アプリケーションのアイデンティティプールの機能です。ID プールが開発者認証に必要とする認証の唯一の証拠は、ID プール開発者の AWS 認証情報です。アイデンティティプールは、この認証フローで提示したデベロッパーとプロバイダーの識別子の有効性に制限を適用しません。

ベストプラクティスとしては、デベロッパープロバイダーは、以下の条件でのみ実装します。

- デベロッパーが認証した認証情報の使用に関する説明責任を作成するには、デベロッパープロバイダーの名前と識別子を設計して、認証ソースを示します。例: "Logins" : {"MyCorp provider" : "[*provider application ID*]"}。
- 有効期限の長いユーザー認証情報は避けてください。[EC2 インスタンスプロファイル](#)や [Lambda 実行ロール](#)などのサービスにリンクされたロールを使用して ID をリクエストするようにサーバー側のクライアントを設定します。
- 同じアイデンティティプールに内部の信頼ソースと外部の信頼ソースを混在させないでください。デベロッパープロバイダーとシングルサインオン (SSO) プロバイダーを別々のアイデンティティプールに追加します。

アクセスコントロールへの属性の使用

アクセスコントロールの属性は、アクセスコントロール (ABAC) の Amazon Cognito アイデンティティプール実装です。ユーザー属性に基づく Amazon Cognito の ID プール経由で AWS リソースへのアクセスを制御するには、IAM ポリシーを使用することができます。これらの属性は、ソーシャル ID プロバイダーおよび企業 ID プロバイダーから収集できます。プロバイダーのアクセストークンと ID トークン、または SAML アサーション内の属性を、IAM 許可ポリシーで参照できるタグにマップできます。

Amazon Cognito ID プールでは、デフォルトのマッピングを選択する、または独自のカスタムマッピングを作成することが可能です。デフォルトのマッピングでは、ユーザー属性の固定セットに基づいた IAM ポリシーを記述できます。カスタムマッピングでは、IAM 許可ポリシーで参照されるユーザー属性のカスタムセットを選択できます。Amazon Cognito コンソールの [Attribute names] (属性名) は、IAM 許可ポリシーで参照される [Tag key for principal] (プリンシパルのタグキー) にマップされます。

例えば、無料と有料のメンバーシップがあるメディアストリーミングサービスを所有しているとしましょう。メディアファイルを Amazon S3 に保存し、無料タグまたはプレミアムタグでそれらをタ

グ付けします。アクセスコントロールの属性を使用して、ユーザーのプロファイルの一部であるユーザーメンバーシップレベルに基づいて、無料コンテンツと有料コンテンツへのアクセスを許可できます。メンバーシップ属性は、IAM 許可ポリシーに渡されるプリンシパルのタグキーにマップできます。そうすることで、単一の許可ポリシーを作成し、メンバーシップレベルの値とコンテンツファイルのタグに基づいて、プレミアムコンテンツへのアクセスを条件的に許可することができます。

トピック

- [Amazon Cognito の ID プールを用いたアクセスコントロールへの属性の使用](#)
- [アクセスコントロールポリシーへの属性の使用例](#)
- [アクセスコントロールの属性をオフにする \(コンソール\)](#)
- [デフォルトのプロバイダーマッピング](#)

属性を使用したアクセスの制御には、いくつかのメリットがあります。

- アクセスコントロールに属性を使用すると、許可の管理がより効率的になります。異なるジョブ機能のために複数のポリシーを作成するのではなく、ユーザー属性を使用する基本的な許可ポリシーを作成できます。
- アプリケーションのリソースやユーザーを追加または削除するたびにポリシーを更新する必要がなくなる。許可ポリシーは、一致するユーザー属性を持つユーザーのみにアクセス権を付与します。例えば、ユーザーの役職に基づいて、特定の S3 バケットへのアクセスを制御する必要がある場合があります。この場合、定義された役職のユーザーのみにこれらのファイルへのアクセスを許可する許可ポリシーを作成できます。詳細については、「[IAM チュートリアル: ABAC で SAML セッションタグを使用する](#)」を参照してください。
- 属性の値に基づいて許可を許可または拒否するポリシーに、プリンシパルタグとしてこれらの属性を渡すことができる。

Amazon Cognito の ID プールを用いたアクセスコントロールへの属性の使用

アクセスコントロールに属性を使用する前に、以下の前提条件を満たしていることを確認します。

- [AWS アカウント](#)
- [ユーザープール](#)
- [ID プール](#)
- [SDKE の設定](#)

- [統合された ID プロバイダー](#)
- [認証情報](#)

アクセスコントロールに属性を使用するには、データソースとして設定したクレームによって、選択したタグキーの値が設定されます。Amazon Cognito はタグキーと値をユーザーのセッションに適用します。IAM ポリシーでは、`${aws:PrincipalTag/tagkey}` 条件からユーザーのアクセスを評価できます。IAM は、ユーザーのタグの値をポリシーに照らして評価します。

ユーザーに認証情報を渡したい IAM ロールを準備する必要があります。これらのロールの信頼ポリシーで、Amazon Cognito がユーザーのロールを引き継ぐことを許可されている必要があります。アクセスコントロール用の属性については、Amazon Cognito がユーザーの一時セッションにプリンシパルタグを適用することも許可しなければなりません。アクション [AssumeRoleWithWebIdentity](#) を使って、ロールを引き受ける権限を付与します。権限のみのアクション `sts:TagSession` を使って、ユーザーのセッションにタグを付ける権限を付与します。詳細については、「AWS Identity and Access Management ユーザーガイド」の「[AWS Security Token Service でのタグ付けの規則](#)」を参照してください。たとえば、`sts:AssumeRoleWithWebIdentity` と `sts:TagSession` 権限を Amazon Cognito サービスプリンシパル `cognito-identity.amazonaws.com` に付与する信頼ポリシーの例については、「[アクセスコントロールポリシーへの属性の使用例](#)」を参照してください。

コンソールでアクセスコントロールの属性を無効にするには

1. [Amazon Cognito コンソール](#) にサインインし、アイデンティティプールを選択します。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. ID プロバイダーを見つけます。編集する ID プロバイダーを選択します。新しい IdP を追加する場合は、[ID プロバイダーを追加] を選択します。
4. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] で [編集] を選択します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. `sub` および `aud` クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
5. [変更を保存] を選択します。

アクセスコントロールポリシーへの属性の使用例

ある企業の法務部門の従業員が、その部門に属しており、セキュリティレベルで分類されているバケット内のすべてのファイルをリストする必要があるというシナリオを考えてみましょう。この従業員が ID プロバイダーから取得するトークンには、以下のクレームが含まれているとします。

クレーム

```
{ .
  .
  "sub" : "57e7b692-4f66-480d-98b8-45a6729b4c88",
  "department" : "legal",
  "clearance" : "confidential",
  .
  .
}
```

これらの属性は、タグにマップして、IAM 許可ポリシーでプリンシパルタグとして参照できます。これで、ID プロバイダー側でユーザープロファイルを変更することによって、アクセスを管理できるようになります。または、ポリシー自体を変更せずに、名前またはタグを使用することによって、リソース側の属性を変更することもできます。

以下の許可ポリシーは、以下の 2 つを行います。

- ユーザーの部門名に一致するプレフィックスで終わるすべての S3 バケットへのリストアクセスを許可します。
- ファイルのクリアランスタグがユーザーのクリアランス属性と一致している限り、これらのバケット内のファイルに対する読み取りアクセスを許可する。

アクセス許可ポリシー

JSON

```
{
  "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}"
        }
      }
    }
  ]
}

```

信頼ポリシーは、誰がこのロールを引き受けられるのかを決定します。信頼関係ポリシーは、アクセスを許可するための `sts:AssumeRoleWithWebIdentity` と `sts:TagSession` の使用を可能にします。これは、ポリシーをユーザーが作成したアイデンティティプールに制限する条件を追加し、それが認証されたロール用であることを確実にします。

信頼ポリシー

JSON

```

{
  "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"
      }
    }
  }
}
```

アクセスコントロールの属性をオフにする (コンソール)

アクセスコントロールの属性を非アクティブにするには、以下の手順に従います。

コンソールでアクセスコントロールの属性を非アクティブにするには

1. [Amazon Cognito コンソール](#)にサインインし、アイデンティティプールを選択します。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. ID プロバイダーを見つけます。編集する ID プロバイダーを選択します。
4. [アクセスコントロールの属性] で、[編集] を選択します。
5. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
6. [変更を保存] を選択します。

デフォルトのプロバイダーマッピング

以下の表には、Amazon Cognito がサポートする認証プロバイダーのデフォルトマッピング情報が記載されています。

プロバイダー	トークンタイプ	プリンシパルタグの値	例
Amazon Cognito ユーザープール	ID トークン	aud (クライアント ID) および sub (ユーザー ID)	"6jk8ltokc7ac9esjrtg9q572f"、"57e7b692-4f66-480d-98b8-45a6729b4c88"
Facebook	アクセストークン	aud(app_id)、sub(user_id)	「492844718097981」、 「112177216992379」
Google	ID トークン	aud (クライアント ID) および sub (ユーザー ID)	「620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.apps.googleusercontent.com」、 「109220063452404746097」
SAML	アサーション	「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 トークン	aud (クライアント ID) および sub (ユーザー ID)	「com.amazonaws.ec2-54-80-172-243.compute-1.client」、 「001968.a6ca34e9c1e742458a26cf8005854be9.0733」

プロバイダー	トークンタイプ	プリンシパルタグの値	例
Amazon	アクセストークン	aud (Amzn Dev Ac のクライアント ID)、user_id (ユーザー ID)	「amzn1.ap plication-oa2-clie nt.9d70d9382d34461 08aee3dd7 63a0fa6」、 「amzn1.account.Agh nifjqmfsbg3g6xcPVB 35orqaa」
標準 OIDC プロバイダー	ID トークンとアクセストークン	aud (client_id として)、sub (ユーザー ID として)	「620493171733- eebk7c0hcp5lj 3e1tlqp1gntt3k0rnc v.apps.googleuserc ontent.com」、 「10922006345240 4746097」
Twitter	アクセストークン	aud (アプリID; アプリシークレット)、サブ (ユーザー ID)	"DfwifTtKEX1FiIBRn OTIR0CFK; Xgj5xb8xlrIVCPjXgL ldkW7fXmw cJJrFvnoK9gwZkLexo 1y5z1", "1269003884 292222976"
DevAuth	マップ	該当なし	「tag1」、 「tag2」

Note

デフォルト属性マッピングオプションは、[Tag Key for Principal] (プリンシパルのタグキー) と [Attribute] (属性) の名前に自動で入力されます。デフォルトマッピングを変更することはできません。

ロールベースアクセスコントロールの使用

Amazon Cognito ID プールは、認証されたユーザーに、AWS リソースにアクセスするための権限が制限された一時的な認証情報のセットを割り当てます。各ユーザーの許可は、作成された [IAM ロール](#) を介して制御されます。ユーザーの ID トークンのクレームに基づいて、各ユーザーのロールを選択するルールを定義できます。認証されたユーザー用のデフォルトのロールを定義できます。また、認証されていないゲストユーザーに対して、権限が制限された個別の IAM ロールを定義することもできます。

ロールマッピング用のロールの作成

各ロールに適切な信頼ポリシーを追加し、Amazon Cognito が ID プール内の認証されたユーザー限定でそのポリシーを引き受けるようにすることが重要です。このような信頼ポリシーの例を示します。

JSON

```
{
  "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"
        }
      }
    }
  ]
}
```

このポリシーでは、cognito-identity.amazonaws.com のフェデレーテッドユーザー (OpenID Connect トークンの発行者) が、このロールを引き受けることができます。さらに、ポリシーでは、ID プールに一致させるためにトークンの aud が制限されます (この場合は ID プール ID)。最後に、このポリシーは、Amazon Cognito によって発行されたトークンの複数の値からなる amr クレームの配列メンバーの 1 つを指定します。GetOpenIdToken API アクションは authenticated 値を含みます。

pass-role 許可の付与

ユーザーが、ID プールでユーザーの既存のアクセス許可を超えるアクセス許可でロールを設定できるようにするには、set-identity-pool-roles API にロールを渡すための iam:PassRole アクセス許可をそのユーザーに付与します。例えば、ユーザーが Amazon S3 に書き込むことはできなくても、ID プールでユーザーが設定する IAM ロールが Amazon S3 への書き込み許可を付与するという場合、ユーザーは iam:PassRole 許可がそのロールに付与されている場合にのみ、このロールを設定できます。次のサンプルポリシー例は、iam:PassRole 権限を付与する方法を示しています。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/myS3WriteAccessRole"
      ]
    }
  ]
}
```

このサンプルポリシーでは、iam:PassRole ロールに myS3WriteAccessRole アクセス権限を付与します。このロールは、それ自体の Amazon リソースネーム (ARN) を使用して指定します。また、このポリシーをユーザーにアタッチする必要があります。詳細については、「[管理ポリシーの使用](#)」を参照してください。

Note

Lambda 関数はリソーススペースのポリシーを使用し、このポリシーは Lambda 関数自体に直接アタッチされます。Lambda 関数を呼び出すルールを作成するときはロールを渡さないため、ルールを作成するユーザーに `iam:PassRole` 許可は必要ありません。Lambda 関数の認可に関する詳細については、「[アクセス許可の管理: Lambda 関数ポリシーを使用する](#)」を参照してください。

ユーザーにロールを割り当てるためのトークンの使用

Amazon Cognito ユーザープール経由でログインするユーザーの場合、ユーザープールによって割り当てられた ID トークンにロールを渡すことができます。これらのロールは、ID トークンの以下のクレームに表示されます。

- `cognito:preferred_role` クレームはロール ARN です。
- `cognito:roles` クレームは、許可されたロール ARN のセットを含むカンマ区切りの文字列です。

クレームは以下のように設定されます。

- `cognito:preferred_role` クレームは、最高 (最も低い) Precedence 値のグループのロールに設定されます。許可されているロールが 1 つだけである場合、`cognito:preferred_role` はそのロールに設定されます。複数のロールがあり、いずれの単一のロールにも最良の優先順位がない場合、このクレームは設定されません。
- `cognito:roles` クレームは、少なくとも 1 つのロールがある場合に設定されます。

ロールの割り当てにトークンを使用するときに、ユーザーに割り当てることができる複数のロールがある場合は、Amazon Cognito ID プール (フェデレーティッド ID) が以下のようにロールを選択します。

- [GetCredentialsForIdentity](#) CustomRoleArn パラメータが設定されており、これが `cognito:roles` クレームのロールに一致する場合は、このパラメータを使用します。このパラメータが `cognito:roles` のロールと一致しない場合は、アクセスを拒否します。
- `cognito:preferred_role` クレームが設定されている場合は、それを使用します。

- `cognito:preferred_role` クレームが設定されておらず、`cognito:roles` クレームが設定されており、`GetCredentialsForIdentity` に対する呼び出しで `CustomRoleArn` が指定されていない場合、割り当てるロールの決定には、コンソールの [ロールの解決] 設定、または `AmbiguousRoleResolution` フィールド ([SetIdentityPoolRoles](#) API の `RoleMappings` パラメータ内) が使用されます。

ルールベースのマッピングを使用してユーザーにロールを割り当てる

ルールは、IAM ロールに ID プロバイダトークンからのクレームをマップすることを許可します。

各ルールは、トークンクレーム (Amazon Cognito ユーザープールからの ID トークンのユーザー属性など)、一致タイプ、値、および IAM ロールを指定します。一致タイプは、`Equals`、`NotEqual`、`StartsWith`、または `Contains` となります。クレームに一致する値をユーザーが持っている場合、そのユーザーは、認証情報を取得するときにそのロールを引き受けることができます。例えば、`custom:dept` カスタム属性値が `Sales` のユーザーに特定の IAM ロールを割り当てるルールを作成できます。

Note

ルールの設定で、カスタム属性は標準属性と区別するために、`custom:` プレフィックスを必要とします。

`CustomRoleArn` が順序を上書きするように指定されている場合を除き、ルールは順序に従って評価され、一致する最初のルールの IAM ロールが使用されます。Amazon Cognito ユーザープールでのユーザー属性の詳細については、「[ユーザー属性の操作](#)」を参照してください。

ID プール (フェデレーティッド ID) コンソールでは、認証プロバイダー用に複数のルールを設定できます。ルールは順序に従って適用されます。ルールをドラッグして、それらの順序を変更できます。一致する最初のルールが優先されます。一致タイプが `NotEqual` で、クレームが存在しない場合、ルールは評価されません。一致するルールがない場合は、[ロールの解決] 設定が [デフォルトの認証されたロールを使用する] または [リクエストの拒否] に適用されます。

API および CLI では、[SetIdentityPoolRoles](#) API の `RoleMappings` パラメータで指定されている [RoleMapping](#) タイプの `AmbiguousRoleResolution` フィールドに一致するルールがない場合に、割り当てられるロールを指定できます。

Amazon Cognito コンソールで ID プロバイダーにルールベースのマッピングを追加するには、IdP を追加または更新し、[ロールの選択] で [ルールを使用してロールを選択する] を選択します。そこから、プロバイダーのクレームを IAM ロールにマッピングするルールを追加できます。

[RoleMapping](#) タイプの RulesConfiguration フィールドを使用して、AWS CLI または API で ID プロバイダーのルールベースのマッピングを設定できます。このフィールドは、[SetIdentityPoolRoles](#) API の RoleMappings パラメータで指定できます。

たとえば、次の AWS CLI コマンドは、OIDC IdP によって認証された Sacramento ロケーションの `arn:aws:iam::123456789012:role/Sacramento_team_S3_admin` ユーザーにロールを割り当てるルールを追加します `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
```

role-mapping.json の内容:

```
{
  "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"
          }
        ]
      }
    }
  }
}
```

ID プールに対して設定された各ユーザープールまたはその他の認証プロバイダーの場合、最大 25 のルールを作成できます。この制限は調整できません。詳細については、「[Amazon Cognito のクォータ](#)」を参照してください。

ルールベースのマッピングで使用するトークンクレーム

Amazon Cognito

Amazon Cognito ID トークンは JSON Web トークン (JWT) として表されます。トークンには、認証ユーザーの ID に関するクレームが含まれます。たとえば、name、family_name、phone_number などです。標準クレームに関する詳細については、「[OpenID Connect 仕様](#)」を参照してください。標準クレームとは別に、以下のような Amazon Cognito に固有の追加のクレームがあります。

- cognito:groups
- cognito:roles
- cognito:preferred_role

Amazon

以下のクレームとそのクレームに使用できる値を、Login with Amazon で使用できます。

- iss: www.amazon.com
- aud: App Id
- sub: Login with Amazon トークンからの sub

Facebook

以下のクレームとそのクレームに使用できる値を、Facebook で使用できます。

- iss: graph.facebook.com
- aud: App Id
- sub: Facebook トークンからの sub

Google

Google トークンには、[OpenID Connect 仕様](#)からの標準クレームが含まれます。OpenID トークンのクレームはすべて、ルールベースのマッピングに使用できます。Google トークンで使用できるクレームについては、Google の [OpenID Connect](#) サイトを参照してください。

Apple

Apple トークンには、[OpenID Connect specification](#) からの標準クレームが含まれます。Apple トークンから入手できるクレームの詳細については、Apple のドキュメントの「[Apple でサインインを使用してユーザーを認証する](#)」を参照してください。email は Apple のトークンには必ずしも含まれているわけではありません。

OpenID

Open ID トークンのクレームはすべて、ルールベースのマッピングに使用できます。標準クレームに関する詳細については、「[OpenID Connect 仕様](#)」を参照してください。使用できる追加クレームについては、ご使用の OpenID プロバイダーのドキュメントを参照してください。

SAML

クレームは受信した SAML アサーションから分析されます。SAML アサーションで利用可能なすべてのクレームは、ルールベースのマッピングで使用できます。

ルールベースのアクセスコントロールのベストプラクティス

Important

ルールにマッピングするクレームがエンドユーザーによって変更できる場合、いずれのエンドユーザーもルールを引き受け、それに応じてポリシーを設定できます。エンドユーザーによって直接設定できないクレームのみを、昇格されたアクセス権限のあるルールにマッピングします。Amazon Cognito ユーザープールで、各ユーザー属性にアプリごとの読み取りおよび書き込み許可を設定できます。

Important

Amazon Cognito ユーザープールのグループにルールを設定する場合、これらのルールはユーザーの ID トークンを通じて渡されます。これらのルールを使用するには、ID プールの認証ルールの選択に対して [トークンからルールを選択する] を設定する必要があります。トークンから正しいルールを判断できない場合のデフォルト動作を指定するには、コンソールの [Role resolution] (ルールの解決) 設定と、[SetIdentityPoolRoles](#) API の RoleMappings パラメータを使用できます。

認証情報の取得

Amazon Cognito を使用して、ユーザーが AWS リソースにアクセスできるように、権限が制限された一時的な認証情報をアプリケーションに配信できます。このセクションでは、認証情報を取得する方法と、ID プールから Amazon Cognito ID を取得する方法について説明します。

Amazon Cognito は、認証されている ID と認証されていない ID の両方をサポートします。認証されていないユーザーは ID が検証されないため、このロールはアプリケーションのゲストユーザーに適切です。または、ユーザーの ID が検証されているかどうかは重要ではない場合に適切です。認証されているユーザーは、アプリケーションにログイン際にサードパーティーの ID プロバイダーまたはユーザープールを通じて ID が検証されます。リソースの許可の範囲を適切に設定し、認証されていないユーザーからのアクセスを許可しないようにします。

Amazon Cognito ID は認証情報ではありません。これらは、AWS Security Token Service () のウェブ ID フェデレーションサポートを使用して認証情報と交換されます AWS STS。アプリケーションユーザーによる AWS 認証情報の取得に推奨される方法は、`AWS.CognitoIdentityCredentials` の使用です。認証情報オブジェクトの ID は、を使用して認証情報と交換されます AWS STS。

Note

2015 年 2 月より前に作成したアイデンティティプールの場合で、ロールをパラメータとせずに `AWS.CognitoIdentityCredentials` コンストラクターを使用するには、再度そのロールと ID プールの関連付けを行う必要があります。これを実行するには、[Amazon Cognito コンソール](#)を開き、[Manage identity pools] (ID プールの管理) をクリックしてから ID プールを選択し、[Edit identity Pool] (ID プールの編集) をクリックします。さらに、認証済みと認証済みでないロールを指定した上で変更を保存します。

ウェブ ID 認証情報プロバイダーは、AWS SDK のデフォルトの認証情報プロバイダーチェーンの一部です。SDK または の AWS ローカル config ファイルに ID プールトークンを設定するには AWS CLI、`web_identity_token_file` プロファイルエントリを追加します。「SDK およびツールリファレンスガイド」の「[ロール認証情報プロバイダーを引き受ける](#)」を参照してください。AWS SDKs

SDK にウェブ ID 認証情報を入力する方法の詳細については、SDK デベロッパーガイドを参照してください。最良の結果を得るには、に組み込まれている ID プール統合を使用してプロジェクトを開始します AWS Amplify。

AWS ID プールで認証情報を取得および設定するための SDK リソース

- Amplify Dev Center の [アイデンティティプールフェデレーション](#) (Android)
- Amplify Dev Center の [アイデンティティプールフェデレーション](#) (iOS)
- AWS SDK for JavaScript デベロッパーガイドの [Amazon Cognito ID を使用してユーザーを認証する](#)
- AWS SDK for .NET デベロッパーガイドの [Amazon Cognito 認証情報プロバイダー](#)
- デ AWS SDK for Go デベロッパーガイドの [「認証情報をプログラムで指定する」](#)
- AWS SDK for Java 2.x デベロッパーガイドの [コードで一時的な認証情報を指定する](#)
- AWS SDK for PHP デベロッパーガイドの [assumeRoleWithWebIdentityCredentialProvider](#) プロバイダー
- AWS SDK for Python (Boto3) ドキュメントの [「ウェブ ID プロバイダーの役割を引き受ける」](#)
- デ AWS SDK for Rust デベロッパーガイドの [認証情報とデフォルトリージョンの指定](#)

以下のセクションでは、一部の AWS SDKs のコード例を示します。

Android

Amazon Cognito を使用して、ユーザーが AWS リソースにアクセスできるように、権限が制限された一時的な認証情報をアプリケーションに配信できます。Amazon Cognito は、認証されている ID と認証されていない ID の両方をサポートします。アプリに AWS 認証情報を提供するには、以下の手順に従います。

Android アプリで Amazon Cognito ID プールを使用するには、を設定します AWS Amplify。詳細については、Amplify Dev Center の「[認証](#)」を参照してください。

Amazon Cognito アイデンティティの取得

認証されていないユーザーを許可している場合は、エンドユーザーの固有 Amazon Cognito 識別子 (アイデンティティ ID) をただちに取得できます。ユーザーを認証している場合は、認証情報プロバイダーでログイントークンを設定した後で、アイデンティティ ID を取得できます。

```
String identityId = credentialsProvider.getIdentityId();
Log.d("LogTag", "my ID is " + identityId);
```

Note

アプリケーションのメインスレッドで、`getIdentityId()`、`refresh()`、または `getCredentials()` を呼び出さないでください。Android 3.0 (API レベル 11) 時点では、メインアプリケーションスレッドでネットワーク I/O を実行すると、アプリで自動的にエラーが発生し、[NetworkOnMainThreadException](#) がスローされます。AsyncTask を使用して、バックグラウンドスレッドにコードを移動する必要があります。詳細については、「[Android のドキュメント](#)」を参照してください。`getCachedIdentityId()` を呼び出して ID を取得することもできますが、この操作が可能なのは、既に ID がローカルにキャッシュされている場合のみです。それ以外の場合、メソッドは null を返します。

iOS - Objective-C

Amazon Cognito を使用して、ユーザーが AWS リソースにアクセスできるように、権限が制限された一時的な認証情報をアプリケーションに配信できます。Amazon Cognito ID プールは、認証された ID と認証されていない ID の両方をサポートします。アプリに AWS 認証情報を提供するには、次の手順を実行します。

iOS アプリで Amazon Cognito ID プールを使用するには、を設定します AWS Amplify。詳細については、Amplify Dev Center の「[Swift 認証](#)」と「[Flutter 認証](#)」を参照してください。

Amazon Cognito アイデンティティの取得

エンドユーザー用の固有 Amazon Cognito 識別子 (アイデンティティ ID) は、認証されていないユーザーを許可している場合はただちに取得、またはユーザーを認証している場合は、認証情報プロバイダーでログイントークンを設定した後で取得することができます。

```
// 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 は非同期呼び出しです。アイデンティティ ID が既にプロバイダーで設定されている場合は、credentialsProvider.identityId を呼び出して (ローカルにキャッシュされた) その ID を取得できます。ただし、プロバイダーでアイデンティティ ID が設定されていない場合、credentialsProvider.identityId を呼び出すと nil が返されます。詳細については、[Amplify iOS SDK リファレンス](#)を参照してください。

iOS - Swift

Amazon Cognito を使用して、権限が制限された一時的な認証情報をアプリケーションに配信し、ユーザーが AWS リソースにアクセスできるようにします。Amazon Cognito は、認証されている ID と認証されていない ID の両方をサポートします。アプリに AWS 認証情報を提供するには、以下の手順に従います。

iOS アプリで Amazon Cognito ID プールを使用するには、を設定します AWS Amplify。詳細については、Amplify Dev Center の「[Swift 認証](#)」を参照してください。

Amazon Cognito アイデンティティの取得

エンドユーザー用の固有 Amazon Cognito 識別子 (アイデンティティ ID) は、認証されていないユーザーを許可している場合はただちに取得、またはユーザーを認証している場合は、認証情報プロバイダーでログイントークンを設定した後で取得することができます。

```
// 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 は非同期呼び出しです。アイデンティティ ID が既にプロバイダーで設定されている場合は、credentialsProvider.identityId を呼び出して (ローカルにキャッシュされた) その ID を取得できます。ただし、プロバイダーでアイデンティティ ID が設定されていない場合、credentialsProvider.identityId を呼び出すと nil が返されます。詳細については、[Amplify iOS SDK リファレンス](#)を参照してください。

JavaScript

ID プールをまだ作成して AWS.CognitoIdentityCredentials を使用する前に [Amazon Cognito コンソール](#)で ID プールを作成してください。

ID プロバイダーで ID プールを設定すると、AWS.CognitoIdentityCredentials を使用してユーザーを認証できます。AWS.CognitoIdentityCredentials を使用するようにアプリケーションを設定するには、credentials またはサービス別の設定の AWS.Config プロパティを設定します。次の例では AWS.Config を使用しています。

```
// 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;
```

```
});
```

オプションの Logins プロパティは、ID プロバイダー名の ID トークンへのマッピングです。ID プロバイダーからのトークンの取得方法は、使用するプロバイダーによって異なります。たとえば、Facebook を ID プロバイダーとして使用する場合は、[FB.loginFacebook SDK の関数](#)を使用して ID プロバイダートークンを取得します。

```
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.');
```

```
  }
});
```

Amazon Cognito アイデンティティの取得

エンドユーザー用の固有 Amazon Cognito 識別子 (アイデンティティ ID) は、認証されていないユーザーを許可している場合はただちに取得、またはユーザーを認証している場合は、認証情報プロバイダーでログイントークンを設定した後で取得することができます。

```
var identityId = AWS.config.credentials.identityId;
```

Unity

Amazon Cognito を使用して、ユーザーが AWS リソースにアクセスできるように、権限が制限された一時的な認証情報をアプリケーションに配信できます。Amazon Cognito は、認証されている ID と認証されていない ID の両方をサポートします。アプリに AWS 認証情報を提供するには、以下の手順に従います。

[AWS SDK for Unity](#) も、[SDK for .NET](#) の一部になりました。で Amazon Cognito の使用を開始するには SDK for .NET、AWS SDK for .NET デベロッパーガイドの[Amazon Cognito 認証情報プロバイダー](#)」を参照してください。または、でアプリケーションを構築するためのオプションについては、[「Amplify 開発センター」](#)を参照してください AWS Amplify。

Amazon Cognito アイデンティティの取得

エンドユーザー用の固有 Amazon Cognito 識別子 (アイデンティティ ID) は、認証されていないユーザーを許可している場合はただちに取得、またはユーザーを認証している場合は、認証情報プロバイダーでログイントークンを設定した後で取得することができます。

```
credentials.GetIdentityIdAsync(delegate(AmazonCognitoIdentityResult<string> result) {  
    if (result.Exception != null) {  
        //Exception!  
    }  
    string identityId = result.Response;  
});
```

Xamarin

Amazon Cognito を使用して、権限が制限された一時的な認証情報をアプリケーションに配信し、ユーザーが AWS リソースにアクセスできるようにします。Amazon Cognito は、認証されている ID と認証されていない ID の両方をサポートします。アプリに AWS 認証情報を提供するには、以下の手順に従います。

[AWS SDK for Xamarin](#) も、[SDK for .NET](#) の一部になりました。で Amazon Cognito の使用を開始するには SDK for .NET、AWS SDK for .NET デベロッパーガイドの[Amazon Cognito 認証情報プロバイダー](#)」を参照してください。または、でアプリケーションを構築するためのオプションについては、「[Amplify 開発センター](#)」を参照してください AWS Amplify。

Note

注: 2015 年 2 月より前に作成した ID プールで、ロールをパラメータとせずに、このコンストラクターを使用するには、再度そのロールと ID プールの関連付けを行う必要があります。これを実行するには、[Amazon Cognito コンソール](#)を開き、[Manage identity pools] (ID プールの管理) をクリックしてから ID プールを選択し、[Edit identity Pool] (ID プールの編集) をクリックします。さらに、認証済みと認証済みでないロールを指定した上で変更を保存します。

Amazon Cognito アイデンティティの取得

エンドユーザー用の固有 Amazon Cognito 識別子 (アイデンティティ ID) は、認証されていないユーザーを許可している場合はただちに取得、またはユーザーを認証している場合は、認証情報プロバイダーでログイントークンを設定した後で取得することができます。

```
var identityId = await credentials.GetIdentityIdAsync();
```

一時的な認証情報 AWS のサービス を使用した へのアクセス

アイデンティティプールによる認証が成功した結果が、一連の AWS 認証情報です。これらの認証情報を使用すると、アプリケーションは IAM 認証で保護された AWS リソースにリクエストを行うことができます。ID プール API オペレーションにアクセスするためにアプリケーションに追加できるさまざまな AWS SDKs を使用すると、一時的な認証情報を生成する認証されていない API リクエストを作成できます。その後、他の SDKs AWS のサービス をクライアントに追加し、それらの一時的な認証情報を使用してリクエストに署名できます。一時的な認証情報ロールに付与された IAM アクセス許可は、他のサービスにリクエストしたオペレーションを許可する必要があります。

Amazon Cognito 認証情報プロバイダーを設定して AWS 認証情報を取得したら、AWS のサービスクライアントを作成します。AWS SDK ドキュメントの例を以下に示します。

AWS クライアントを作成するための SDK リソース

- AWS SDK for C++ デベロッパーガイドの[AWS クライアント設定](#)
- AWS SDK for Go デベロッパーガイドの「[での AWS SDK for Go V2 の使用 AWS のサービス](#)」
- AWS SDK for Java 2.x デベロッパーガイドの「[HTTP クライアントの設定](#)」
- AWS SDK for JavaScript デベロッパーガイドの「[サービスオブジェクトの作成と呼び出し](#)」
- AWS SDK for Python (Boto3) ドキュメントでの[クライアントの作成](#)
- デ AWS SDK for Rust ベロッパーガイドの「[サービスクライアントの作成](#)」
- AWS SDK for Swift デベロッパーガイドの[クライアントの使用](#)

以下のスニペットは Amazon DynamoDB クライアントを初期化します。

Android

Android アプリで Amazon Cognito ID プールを使用するには、を設定します AWS Amplify。詳細については、Amplify Dev Center の「[認証](#)」を参照してください。

```
// Create a service client with the provider
AmazonDynamoDB client = new AmazonDynamoDBClient(credentialsProvider);
```

認証情報プロバイダーは Amazon Cognito と通信し、認証されたユーザーと認証されていないユーザーの一意の識別子と、AWS Mobile SDK の一時的で制限された特権 AWS 認証情報の両方を取得

します。取得された認証情報は 1 時間有効で、有効期限が切れるとプロバイダーによって更新されます。

iOS - Objective-C

iOS アプリで Amazon Cognito ID プールを使用するには、を設定します AWS Amplify。詳細については、Amplify Dev Center の「[Swift 認証](#)」と「[Flutter 認証](#)」を参照してください。

```
// 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];
```

認証情報プロバイダーは Amazon Cognito と通信し、認証されたユーザーと認証されていないユーザーの一意の識別子と、AWS Mobile SDK の一時的で制限された特権 AWS 認証情報の両方を取得します。取得された認証情報は 1 時間有効で、有効期限が切れるとプロバイダーによって更新されます。

iOS - Swift

iOS アプリで Amazon Cognito ID プールを使用するには、を設定します AWS Amplify。詳細については、Amplify Dev Center の「[Swift 認証](#)」を参照してください。

```
// 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")
```

認証情報プロバイダーは Amazon Cognito と通信し、認証されたユーザーと認証されていないユーザーの一意の識別子と、AWS Mobile SDK の一時的で制限された特権 AWS 認証情報の両方を取得します。取得された認証情報は 1 時間有効で、有効期限が切れるとプロバイダーによって更新されます。

JavaScript

```
// Create a service client with the provider
var dynamodb = new AWS.DynamoDB({region: 'us-west-2'});
```

認証情報プロバイダーは Amazon Cognito と通信し、認証されたユーザーと認証されていないユーザーの一意の識別子と、AWS Mobile SDK の一時的な制限付き特権 AWS 認証情報の両方を取得します。取得された認証情報は 1 時間有効で、有効期限が切れるとプロバイダーによって更新されます。

Unity

[AWS SDK for Unity](#) も、[SDK for .NET](#) の一部になりました。で Amazon Cognito の使用を開始するには SDK for .NET、「AWS SDK for .NET デベロッパーガイド」の[Amazon Cognito 認証情報プロバイダー](#)」を参照してください。または、でアプリケーションを構築するためのオプションについては、「[Amplify 開発センター](#)」を参照してください AWS Amplify。

```
// create a service client that uses credentials provided by Cognito
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials, REGION);
```

認証情報プロバイダーは Amazon Cognito と通信し、認証されたユーザーと認証されていないユーザーの一意の識別子と、AWS Mobile SDK の一時的な制限付き特権 AWS 認証情報の両方を取得します。取得された認証情報は 1 時間有効で、有効期限が切れるとプロバイダーによって更新されます。

Xamarin

[AWS SDK for Xamarin](#) も、[SDK for .NET](#) の一部になりました。で Amazon Cognito の使用を開始するには SDK for .NET、「AWS SDK for .NET デベロッパーガイド」の[Amazon Cognito 認証情報プロバイダー](#)」を参照してください。または、でアプリケーションを構築するためのオプションについては、「[Amplify 開発センター](#)」を参照してください AWS Amplify。

```
// create a service client that uses credentials provided by Cognito
var client = new AmazonDynamoDBClient(credentials, REGION)
```

認証情報プロバイダーは Amazon Cognito と通信し、認証されたユーザーと認証されていないユーザーの一意の識別子と、AWS Mobile SDK の一時的な制限付き特権 AWS 認証情報の両方を取得します。取得された認証情報は 1 時間有効で、有効期限が切れるとプロバイダーによって更新されます。

アイデンティティプールのサードパーティー ID プロバイダー

Amazon Cognito ID プールを使用すると、さまざまな外部 ID プロバイダー (IdPs) と統合して、アプリケーションのフェデレーション認証を通じて一時的な AWS 認証情報を提供できます。これらの外

部 IdPs で動作するように ID プールを設定することで、Amazon Cognito ユーザープール、ソーシャルプロバイダー、OIDC プロバイダー、または SAML プロバイダーによる認証を使用して、ユーザーのバックエンド AWS リソースへのアクセスを許可できます。このセクションでは、IdP を設定して Amazon Cognito アイデンティティプールと統合する手順について説明します。

logins プロパティを使用すると、ID プロバイダー (IdP) から受信した認証情報を設定できます。さらに、アイデンティティプールと複数の IdP を関連付けることもできます。例えば、Facebook と Google の両方のトークンを logins プロパティに設定して、一意の Amazon Cognito アイデンティティが両方の IdP のログインに関連付けられるようにすることができます。ユーザーはどちらのアカウントでも認証できますが、Amazon Cognito は同じユーザー ID を返します。

以下の手順では、Amazon Cognito アイデンティティプールによってサポートされている IdP での認証について説明します。

トピック

- [アイデンティティプール IdP として Facebook を設定する](#)
- [アイデンティティプール IdP として Login with Amazon を設定する](#)
- [アイデンティティプール IdP として Google を設定する](#)
- [アイデンティティプール IdP として Apple でサインインを設定する](#)
- [OIDC プロバイダーをアイデンティティプール IdP として設定する](#)
- [SAML プロバイダーをアイデンティティプール IdP として設定する](#)

アイデンティティプール IdP として Facebook を設定する

Amazon Cognito アイデンティティプールは、アプリケーションユーザーにフェデレーション方式の認証を提供するために Facebook と連携します。このセクションでは、IdP として Facebook に対してアプリケーションを登録し、セットアップする方法について説明します。

Facebook のセットアップ

Facebook ユーザーを認証し、Facebook API と統合するには、事前にアプリケーションを Facebook に登録します。

[Facebook デベロッパーポータル](#)は、アプリケーションのセットアップに役立ちます。Amazon Cognito アイデンティティプールに Facebook を統合する前に、次の手順を実行してください。

Note

Amazon Cognito アイデンティティプールフェデレーションは、[Facebook の制限付きログイン](#)と互換性がありません。限定ログインのアクセス許可セットを超えないように iOS 用の Facebook ログインを設定する方法の詳細については、「Meta for Developers」の「[Facebook Login for iOS - Quickstart](#)」を参照してください。

Facebook のセットアップ

1. [Facebook デベロッパーポータル](#)で、Facebook 認証情報を使用してログインします。
2. アプリメニューの Add a New App (新しいアプリを追加する)を選択します。
3. プラットフォームを選択し、クイックスタートプロセスを完了します。

Android

Android アプリと Facebook ログインとの統合に関する詳細情報については、[Facebook 入門ガイド](#)を参照してください。

iOS - Objective-C

iOS Objective-C アプリと Facebook ログインとの統合に関する詳細情報については、[Facebook 入門ガイド](#)を参照してください。

iOS - Swift

iOS Swift アプリと Facebook ログインとの統合に関する詳細情報については、[Facebook 入門ガイド](#)を参照してください。

JavaScript

JavaScript ウェブアプリと Facebook ログインとの統合に関する詳細情報については、[Facebook 入門ガイド](#)を参照してください。

Amazon Cognito アイデンティティプールコンソールでアイデンティティプロバイダーを設定する

以下の手順を使用して、ID プロバイダーを設定します。

Facebook ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Facebook] を選択します。
5. [Meta for Developers](#) で作成した OAuth プロジェクトのアプリ ID を入力します。詳細については、Meta for Developers ドキュメントの「[Facebook ログイン](#)」を参照してください。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
8. [変更を保存] を選択します。

Facebook の使用

Android

Facebook 認証を追加するには、まず [Facebook ガイド](#) に従って、アプリケーションに Facebook SDK を統合します。次に、[\[Login with Facebook\] ボタン](#) を Android ユーザーインターフェイスに追加します。Facebook SDK は、その状態の追跡にセッションオブジェクトを使用します。Amazon Cognito は、このセッションオブジェクトのアクセストークンを使用してユーザーを認証し、一意の識別子を生成し、必要に応じてユーザーに他の AWS リソースへのアクセスを許可します。

Facebook SDK でユーザーを認証したら、Amazon Cognito 認証情報プロバイダーにセッショントークンを追加します。

Facebook SDK 4.0 以降:

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com", AccessToken.getCurrentAccessToken().getToken());
credentialsProvider.setLogins(logins);
```

Facebook SDK 4.0 以前:

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com", Session.getActiveSession().getAccessToken());
credentialsProvider.setLogins(logins);
```

Facebook ログインプロセスにより、その SDK でシングルトンセッションが初期化されます。Facebook セッションオブジェクトには、Amazon Cognito が認証されたエンドユーザーの AWS 認証情報を生成するために使用する OAuth トークンが含まれています。Amazon Cognito は、この特定の Facebook アイデンティティに一致するユーザーの存在についてユーザーデータベースをチェックするためにもこのトークンを使用します。既にユーザーに存在する場合、API は既存の ID を返します。それ以外の場合、API は新しい ID を返します。クライアント SDK は、ローカルデバイスで ID を自動的にキャッシュします。

Note

ログインマップを設定したら、`refresh` または `get` を呼び出し `get` で AWS 認証情報を取得します。

iOS - Objective-C

Facebook 認証を追加するには、まず [Facebook ガイド](#) に従って、アプリケーションに Facebook SDK を統合します。次に、[\[Login with Facebook \(Facebook でログイン\)\]](#) ボタンを ユーザーインターフェイスに追加します。Facebook SDK は、その状態の追跡にセッションオブジェクトを使用します。Amazon Cognito は、このセッションオブジェクトからのアクセストークンを使用してユーザーを認証し、一意の Amazon Cognito ID プール (フェデレーティッド ID) にバインドします。

Amazon Cognito に Facebook アクセストークンを提供するには、[AWSIdentityProviderManager](#) プロトコルを実装します。

logins メソッドを実装するときは、AWSIdentityProviderFacebook を含むディクショナリを返します。以下のコード例にあるように、このディクショナリはキーとして機能し、認証された Facebook ユーザーからの現在のアクセストークンが値として機能します。

```
- (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"}]];
    }
}
```

AWSCognitoCredentialsProvider をインスタンス化するときに、コンストラクターで AWSIdentityProviderManager の値として identityProviderManager を実装するクラスを渡します。詳細については、[AWSCognitoCredentialsProvider](#) リファレンスページに移動し、initWithRegionType:identityPoolId:identityProviderManager を選択してください。

iOS - Swift

Facebook 認証を追加するには、まず [Facebook ガイド](#) に従って、アプリケーションに Facebook SDK を統合します。次に、[\[Login with Facebook \(Facebook でログイン\)\]](#) ボタンを ユーザーインターフェイスに追加します。Facebook SDK は、その状態の追跡にセッションオブジェクトを使用します。Amazon Cognito は、このセッションオブジェクトからのアクセストークンを使用してユーザーを認証し、一意の Amazon Cognito ID プール (フェデレーティッド ID) にバインドします。

Note

Amazon Cognito アイデンティティプールフェデレーションは、[Facebook の制限付きログイン](#)と互換性がありません。限定ログインのアクセス許可セットを超えないように iOS 用の Facebook ログインを設定する方法の詳細については、「Meta for Developers」の「[Facebook Login for iOS - Quickstart](#)」を参照してください。

Amazon Cognito に Facebook アクセストークンを提供するには、[AWSIdentityProviderManager](#) プロトコルを実装します。

logins メソッドを実装するときは、AWSIdentityProviderFacebook を含むディクショナリを返します。以下のコード例にあるように、このディクショナリはキーとして機能し、認証された Facebook ユーザーからの現在のアクセストークンが値として機能します。

```
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"]))
    }
}
```

AWSCognitoCredentialsProvider をインスタンス化するときに、コンストラクターで AWSIdentityProviderManager の値として identityProviderManager を実装するクラスを渡します。詳細については、[AWSCognitoCredentialsProvider](#) リファレンスページに移動し、initWithRegionType:identityPoolId:identityProviderManager を選択してください。

JavaScript

Facebook 認証を追加するには、「[ウェブ用の Facebook ログイン](#)」に従って、ウェブサイトに「Facebook でログイン」ボタンを追加します。Facebook SDK は、その状態の追跡にセッションオブジェクトを使用します。Amazon Cognito は、このセッションオブジェクトのアクセストークンを使用してユーザーを認証し、一意の識別子を生成し、必要に応じてユーザーに他の AWS リソースへのアクセスを許可します。

Facebook SDK でユーザーを認証したら、Amazon Cognito 認証情報プロバイダーにセッショントークンを追加します。

```
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.');
```

```
    }

});
```

Facebook SDK は、Amazon Cognito が認証されたエンドユーザーの AWS 認証情報を生成するために使用する OAuth トークンを取得します。Amazon Cognito は、この特定の Facebook アイデンティティに一致するユーザーの存在についてユーザーデータベースをチェックするためにもこのトークンを使用します。既にユーザーに存在する場合、API は既存の ID を返します。それ以外の場合、新しい ID が返されます。ID はローカルデバイスでクライアント SDK によって自動的にキャッシュされます。

Note

ログインマップを設定した後、refresh または get を呼び出して、認証情報を取得します。コード例については、[JavaScript の README ファイル](#)の「ユースケース 17: ユーザープールと Cognito ID の統合」を参照してください。

Unity

Facebook 認証を追加するには、まず [Facebook ガイド](#) に従って、アプリケーションに Facebook SDK を統合します。Amazon Cognito は FB オブジェクトからの Facebook アクセストークンを使用して、Amazon Cognito アイデンティティに関連付けられた一意のユーザー 識別子を生成します。

Facebook SDK でユーザーを認証したら、Amazon Cognito 認証情報プロバイダーにセッショントークンを追加します。

```
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);
}
```

FB.AccessToken を使用する前に、FB.Login() を呼び出し、FB.IsLoggedIn が true であることを確認してください。

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
    }
});
}
```

アイデンティティプール IdP として Login with Amazon を設定する

Amazon Cognito アイデンティティプールは Login with Amazon と統合して、モバイルアプリケーションとウェブアプリケーションのユーザーにフェデレーション方式の認証を提供します。このセクションでは、ID プロバイダー (IdP) として Login with Amazon を使用してアプリケーションを登録し、セットアップする方法について説明します。

[デベロッパーポータル](#)で Amazon Cognito を使用するための Login with Amazon を設定します。詳細については、Login with Amazon のよくある質問の「[Login with Amazon のセットアップ](#)」を参照してください。

Note

Login with Amazon を Xamarin アプリケーションに統合するには、[Xamarin 入門ガイド](#)に従ってください。

Note

Login with Amazon を Unity プラットフォームでネイティブに統合することはできません。代わりに、ウェブビューを使用して、ブラウザのサインインフローに従います。

Login with Amazon のセットアップ

Login with Amazon の実装

[Amazon デベロッパーポータル](#)では、OAuth アプリケーションをセットアップして ID プールと統合し、Login with Amazon のドキュメントを検索し、SDK をダウンロードできます。[Developer console] (デベロッパーコンソール) を選択し、デベロッパーポータルで Login with Amazon を選択します。アプリケーションのセキュリティプロファイルを作成し、Login with Amazon 認証メカニズムをアプリにビルドできます。Login with Amazon 認証をアプリに統合する方法の詳細については、「[認証情報の取得](#)」を参照してください。

Amazon は、新しいセキュリティプロファイルに対して OAuth 2.0 クライアント ID を発行します。クライアント ID は、セキュリティプロファイルの [Web Settings] (ウェブ設定) タブにあります。アイデンティティプールの Login with Amazon IdP の [アプリケーション ID] フィールドにセキュリティプロファイル ID を入力します。

Note

アイデンティティプールの Login with Amazon IdP の [アプリケーション ID] フィールドにセキュリティプロファイル ID を入力します。これは、クライアント ID を使用するユーザープールとは異なります。

Amazon Cognito コンソールで外部プロバイダーを設定する

Amazon アイデンティティプロバイダー (IdP) を使ってログインを追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Login with Amazon] を選択します。
5. [Login with Amazon](#) で作成した OAuth プロジェクトのアプリ ID を入力します。詳細については、「[Login with Amazon のドキュメント](#)」を参照してください。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。

- c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。

8. [変更を保存] を選択します。

Login with Amazon の使用: Android

Amazon ログインを認証したら、TokenListener インターフェイスの onSuccess メソッドで Amazon Cognito 認証情報プロバイダーにトークンを渡すことができます。コードは次のようになります。

```
@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);
}
```

Login with Amazon の使用: iOS - Objective-C

Amazon のログインを認証したら、AMZNAccessTokenDelegate の requestDidSucceed メソッドで Amazon Cognito 認証情報プロバイダーにトークンを渡すことができます。

```
- (void)requestDidSucceed:(APIResult \*)apiResult {
    if (apiResult.api == kAPIAuthorizeUser) {
        [AIMobileLib getAccessTokenForScopes:[NSArray arrayWithObject:@"profile"]
        withOverrideParams:nil delegate:self];
    }
    else if (apiResult.api == kAPIGetAccessToken) {
        credentialsProvider.logins = @{ @(AWSCognitoLoginProviderKeyLoginWithAmazon):
        apiResult.result };
    }
}
```

Login with Amazon の使用: iOS - Swift

Amazon のログインを認証したら、AMZNAccessTokenDelegate の requestDidSucceed メソッドで Amazon Cognito 認証情報プロバイダーにトークンを渡すことができます。

```
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]
}
}
```

Login with Amazon の使用: JavaScript

ユーザーが Login with Amazon と認証し、ウェブサイトのリダイレクトされると、Login with Amazon access_token がクエリ文字列で提供されます。このトークンを認証情報ログインマップに渡します。

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: 'IDENTITY_POOL_ID',
  Logins: {
    'www.amazon.com': 'Amazon Access Token'
  }
});
```

アイデンティティプール IdP として Google を設定する

Amazon Cognito アイデンティティプールは、モバイルアプリケーションユーザーにフェデレーション方式の認証を提供するために Google と連携します。このセクションでは、ID プロバイダーとして Google に対してアプリケーションを登録し、セットアップする方法について説明します。

Android

Note

アプリが Google を使用していて、複数のモバイルプラットフォームで利用可能になる場合は、[OpenID Connect プロバイダー](#)として設定する必要があります。作成されたすべてのクライアント ID を追加オーディエンス値として追加することで、統合が向上します。Google のクライアント間の ID モデルの詳細については、「[クライアント間の ID](#)」を参照してください。

Google のセットアップ

Android 用 Google サインインを有効化するには、アプリケーション用の Google デベロッパーコンソールプロジェクトを作成する必要があります。

1. [Google 開発者コンソール](#)に移動して、新しいプロジェクトを作成します。
2. [APIs & Services] (API とサービス) を選択し、次に [OAuth consent screen] (OAuth 同意画面) を選択します。Google がプロフィールデータをアプリと共有するための同意を求められたときに、Google がユーザーに表示する情報をカスタマイズします。
3. [Credentials] (認証情報) を選択し、次に[Create credentials] (認証情報の作成) を選択します。[OAuth client ID] (OAuth クライアント ID) を選択します。アプリケーションタイプとして [Android] を選択します。アプリを開発するプラットフォームごとに個別のクライアント ID を作成します。
4. [Credentials] (認証情報) で、[Manage service accounts] (サービスアカウントの管理) を選択します。[Create service account] (サービスアカウントの作成) を選択します。サービスアカウントの詳細を入力し、[Create and continue] (作成して続ける) を選択します。
5. サービスアカウントにプロジェクトへのアクセス権を付与します。アプリが必要とするサービスアカウントへのアクセス権をユーザーに付与します。
6. 新しいサービスアカウントを選択し、[Keys] (キー)、および [Add key] (キーを追加する) を選択します。新しい JSON キーを作成してダウンロードします。

Google デベロッパーコンソールの使用方法の詳細については、Google Cloud ドキュメントの「[プロジェクトの作成と管理](#)」を参照してください。

Google を Android アプリケーションに統合する方法の詳細については、Google ID に関するドキュメントの「[「Google でログイン」でユーザーを認証する](#)」を参照してください。

Google ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Google] を選択します。

5. [Google Cloud プラットフォーム](#)で作成した OAuth プロジェクトのクライアント ID を入力します。詳細については、Google Cloud Platform コンソールヘルプの「[OAuth 2.0 の設定](#)」を参照してください。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
8. [変更を保存] を選択します。

Google の使用

アプリケーションで Google を使用したログインを有効にするには、[Android に関する Google のドキュメント](#)に従います。ユーザーがサインインすると、Google に OpenID Connect 認証トークンを要求します。Amazon Cognito はこのトークンを使用してユーザーを認証し、一意の ID を生成します。

次のサンプルコードは、Google Play サービスから認証トークンを取得する方法を示しています。

```
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

アプリが Google を使用していて、複数のモバイルプラットフォームで利用可能になる場合は、Google を [OpenID Connect プロバイダー](#) とします。作成されたすべてのクライアント ID を追加オーディエンス値として追加することで、統合が向上します。Google のクライアント間の ID モデルの詳細については、「[クライアント間の ID](#)」を参照してください。

Google のセットアップ

iOS 用 Google サインインを有効にするには、アプリケーション用の Google デベロッパーコンソールプロジェクトを作成します。

1. [Google 開発者コンソール](#) に移動して、新しいプロジェクトを作成します。
2. [APIs & Services] (API とサービス) を選択し、次に [OAuth consent screen] (OAuth 同意画面) を選択します。Google がプロフィールデータをアプリと共有するための同意を求められたときに、Google がユーザーに表示する情報をカスタマイズします。
3. [Credentials] (認証情報) を選択し、次に [Create credentials] (認証情報の作成) を選択します。[OAuth client ID] (OAuth クライアント ID) を選択します。アプリケーションタイプとして [iOS] を選択します。アプリを開発するプラットフォームごとに個別のクライアント ID を作成します。
4. [Credentials] (認証情報) で、[Manage service accounts] (サービスアカウントの管理) を選択します。[Create service account] (サービスアカウントの作成) を選択します。サービスアカウントの詳細を入力し、[Create and continue] (作成して続ける) を選択します。
5. サービスアカウントにプロジェクトへのアクセス権を付与します。アプリが必要とするサービスアカウントへのアクセス権をユーザーに付与します。
6. 新しいサービスアカウントを選択します。[Keys] (キー) タブを選択し、[Add key] (キーを追加する) を選択します。新しい JSON キーを作成してダウンロードします。

Google デベロッパーコンソールの使用方法の詳細については、Google Cloud ドキュメントの「[プロジェクトの作成と管理](#)」を参照してください。

iOS アプリへの Google の統合に関する詳細については、[Google ID に関するドキュメント](#)の「iOS 用 Google サインイン」を参照してください。

Google ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Google] を選択します。
5. [Google Cloud プラットフォーム](#)で作成した OAuth プロジェクトのクライアント ID を入力します。詳細については、Google Cloud Platform コンソールヘルプの「[OAuth 2.0 の設定](#)」を参照してください。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。

8. [変更を保存] を選択します。

Google の使用

アプリケーションで Google を使用したログインを有効にするには、[iOS に関する Google のドキュメント](#)に従います。認証に成功すると、OpenID Connect 認証トークンが取得されます。Amazon Cognito はこのトークンを使用してユーザーを認証し、固有識別子を生成します。

認証に成功すると、id_token が含まれる GTM0Auth2Authentication オブジェクトが取得されます。Amazon Cognito はこのオブジェクトを使用してユーザーを認証し、固有識別子を生成します。

```
- (void)finishedWithAuth: (GTM0Auth2Authentication *)auth error: (NSError *) error {
    NSString *idToken = [auth.parameters objectForKey:@"id_token"];
    credentialsProvider.logins = @{ @(AWSCognitoLoginProviderKeyGoogle): idToken };
}
```

iOS - Swift

Note

アプリが Google を使用していて、複数のモバイルプラットフォームで利用可能になる場合は、Google を [OpenID Connect プロバイダー](#)とします。作成されたすべてのクライアント ID を追加オーディエンス値として追加することで、統合が向上します。Google のクライアント間の ID モデルの詳細については、「[クライアント間の ID](#)」を参照してください。

Google のセットアップ

iOS 用 Google サインインを有効にするには、アプリケーション用の Google デベロッパーコンソールプロジェクトを作成します。

1. [Google 開発者コンソール](#)に移動して、新しいプロジェクトを作成します。
2. [APIs & Services] (API とサービス) を選択し、次に [OAuth consent screen] (OAuth 同意画面) を選択します。Google がプロフィールデータをアプリと共有するための同意を求められたときに、Google がユーザーに表示する情報をカスタマイズします。
3. [Credentials] (認証情報) を選択し、次に[Create credentials] (認証情報の作成) を選択します。[OAuth client ID] (OAuth クライアント ID) を選択します。アプリケーションタイプとして

[iOS] を選択します。アプリを開発するプラットフォームごとに個別のクライアント ID を作成します。

4. [Credentials] (認証情報) で、[Manage service accounts] (サービスアカウントの管理) を選択します。[Create service account] (サービスアカウントの作成) を選択します。サービスアカウントの詳細を入力し、[Create and continue] (作成して続ける) を選択します。
5. サービスアカウントにプロジェクトへのアクセス権を付与します。アプリが必要とするサービスアカウントへのアクセス権をユーザーに付与します。
6. 新しいサービスアカウントを選択し、[Keys] (キー)、および [Add key] (キーを追加する) を選択します。新しい JSON キーを作成してダウンロードします。

Google デベロッパーコンソールの使用方法の詳細については、Google Cloud ドキュメントの「[プロジェクトの作成と管理](#)」を参照してください。

iOS アプリへの Google の統合に関する詳細については、[Google ID に関するドキュメント](#)の「iOS 用 Google サインイン」を参照してください。

Amazon Cognito コンソールのホームページで [\[Manage Identity Pools\]](#) (ID プールの管理) をクリックします。

Amazon Cognito コンソールで外部プロバイダーを設定する

1. 外部プロバイダーとして Google を有効にする ID プールの名前を選択します。アイデンティティプールの [Dashboard] (ダッシュボード) ページが表示されます。
2. [ダッシュボード] ページの右上にある、[ID プールの編集] を選択します。[Edit identity pool] (ID プールの編集) ページが表示されます。
3. 下にスクロールし、[Authentication providers] (認証プロバイダー) を選択してセクションを展開します。
4. [Google] タブを選択します。
5. [ロック解除] を選択します。
6. Google から入手した Google Client ID を入力し、[Save Changes] (変更の保存) を選択します。

Google の使用

アプリケーションで Google を使用したログインを有効にするには、[iOS に関する Google のドキュメント](#)に従います。認証に成功すると、OpenID Connect 認証トークンが取得されます。Amazon Cognito はこのトークンを使用してユーザーを認証し、一意の識別子を生成します。

認証が成功すると、`id_token` を含む `GTM0Auth2Authentication` オブジェクトが生成されます。Amazon Cognito はこのトークンを使用してユーザーを認証し、一意の ID を生成します。

```
func finishedWithAuth(auth: GTM0Auth2Authentication!, error: NSError!) {
    if error != nil {
        print(error.localizedDescription)
    }
    else {
        let idToken = auth.parameters.objectForKey("id_token")
        credentialsProvider.logins = [AWSCognitoLoginProviderKey.Google.rawValue:
idToken!]
    }
}
```

JavaScript

Note

アプリが Google を使用していて、複数のモバイルプラットフォームで利用可能になる場合は、Google を [OpenID Connect プロバイダー](#) として設定する必要があります。作成されたすべてのクライアント ID を追加オーディエンス値として追加することで、統合が向上します。Google のクライアント間の ID モデルの詳細については、「[クライアント間の ID](#)」を参照してください。

Google のセットアップ

JavaScript ウェブアプリ用 Google サインインを有効にするには、アプリケーション用の Google デベロッパーコンソールプロジェクトを作成します。

1. [Google 開発者コンソール](#) に移動して、新しいプロジェクトを作成します。
2. [APIs & Services] (API とサービス) を選択し、次に [OAuth consent screen] (OAuth 同意画面) を選択します。Google がプロフィールデータをアプリと共有するための同意を求められたときに、Google がユーザーに表示する情報をカスタマイズします。
3. [Credentials] (認証情報) を選択し、次に [Create credentials] (認証情報の作成) を選択します。[OAuth client ID] (OAuth クライアント ID) を選択します。アプリケーションタイプとして [Web application] (ウェブアプリケーション) を選択します。アプリを開発するプラットフォームごとに個別のクライアント ID を作成します。

4. [Credentials] (認証情報) で、[Manage service accounts] (サービスアカウントの管理) を選択します。[Create service account] (サービスアカウントの作成) を選択します。サービスアカウントの詳細を入力し、[Create and continue] (作成して続ける) を選択します。
5. サービスアカウントにプロジェクトへのアクセス権を付与します。アプリが必要とするサービスアカウントへのアクセス権をユーザーに付与します。
6. 新しいサービスアカウントを選択し、[Keys] (キー)、および [Add key] (キーを追加する) を選択します。新しい JSON キーを作成してダウンロードします。

Google デベロッパーコンソールの使用方法の詳細については、Google Cloud ドキュメントの「[プロジェクトの作成と管理](#)」を参照してください。

Google をウェブアプリに統合する方法の詳細については、Google ID に関するドキュメントの「[Google でサインインする](#)」を参照してください。

Amazon Cognito コンソールで外部プロバイダーを設定する

Google ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Google] を選択します。
5. [Google Cloud プラットフォーム](#)で作成した OAuth プロジェクトのクライアント ID を入力します。詳細については、Google Cloud Platform コンソールヘルプの「[OAuth 2.0 の設定](#)」を参照してください。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。

- ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
8. [変更を保存] を選択します。

Google の使用

アプリケーションで Google を使用したログインを有効にするには、「[ウェブに関する Google のドキュメント](#)」に従います。

認証に成功すると、id_token が含まれるレスポンスオブジェクトが取得されます。Amazon Cognito はこのオブジェクトを使用してユーザーを認証し、固有識別子を生成します。

```
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.
    });
  }
}
```

アイデンティティプール IdP として Apple でサインインを設定する

Amazon Cognito アイデンティティプールは「Apple でサインイン」と統合して、モバイルアプリケーションとウェブアプリケーションのユーザーにフェデレーション方式の認証を提供します。このセクションでは、ID プロバイダー (IdP) として「Apple でサインイン」を使用してアプリケーションを登録し、セットアップする方法について説明します。

認証プロバイダーとして「Apple でサインイン」を ID プールに追加するには、2 つのステップが必要です。まず、アプリケーションで「Apple でサインイン」を統合し、次に ID プールで「Apple でサインイン」を設定します。「Apple でサインイン」の設定に関する最新の情報については、Apple デベロッパー向けドキュメントの「[Apple でサインインの環境を設定する](#)」を参照してください。

「Apple でサインイン」を設定する

IdP として「Apple でサインイン」を設定するには、クライアント ID を受け取るためにアプリケーションを Apple に登録する必要があります。

1. [Apple の開発者アカウント](#)を作成します。
2. Apple 認証情報を使用して[サインイン](#)します。
3. 左のナビゲーションペインで、[Certificates, IDs & Profiles (証明書、ID & プロファイル)] を選択します。
4. 左のナビゲーションペインで、[Identifiers (識別子)] を選択します。
5. [Identifiers (識別子)] ページで、[+] アイコンを選択します。
6. [Register a New Identifier] (新しい識別子の登録) ページで、[App IDs] (アプリ ID)、[Continue] (続行) の順に選択します。
7. [Register an App ID] ページで、次の操作を行います。
 - a. [説明] に、説明を入力します。
 - b. [Bundle ID (バンドル ID)] に、識別子を入力します。このバンドル ID を書き留めておきます。Apple を ID プールのプロバイダーとして設定するには、この値が必要になります。
 - c. [Capabilities] (機能) で、[Sign In with Apple] (Apple でサインイン) を選択してから [Edit] (編集) を選択します。
 - d. [Sign in with Apple: App ID Configuration] (Apple でサインイン: アプリ ID の設定) ページで、アプリの適切な設定を選択します。次に、[Save (保存)] を選択します。
 - e. [続行] をクリックしてください。
8. [Confirm your App ID] ページで、[登録] を選択します。

9. 「Apple でサインイン」をネイティブ iOS アプリケーションと統合する場合は、ステップ 10 に進みます。ステップ 11 は、「Apple JS でサインイン」と統合するアプリケーション用です。
10. [ID] ページで、[App IDs] (アプリケーション ID) メニューを選択し、次に [Services IDs] (サービス ID) を選択します。[+] アイコンを選択します。
11. [Register a New Identifier] (新しい識別子の登録) ページで、[Services IDs] (サービス ID)、[Continue] (続行) の順に選択します。
12. [Register a Services ID] ページで、次の操作を行います。
 - a. [説明] に、説明を入力します。
 - b. [Identifier] に、識別子を入力します。このサービス ID を書き留めておきます。Apple を ID プールのプロバイダーとして設定するには、この値が必要になります。
 - c. [Apple でサインイン] を選択後、[Configure (設定)] を選択します。
 - d. [Web Authentication Configuration] ページで、[Primary App ID] を選択します。[Website URLs] (ウェブサイトの URL) で、[+] アイコンを選択します。[Domains and Subdomains (ドメインとサブドメイン)] で、アプリのドメイン名を入力します。[リターン URL] に、ユーザーが「Apple でサインイン」を介して認証した後、認可によってユーザーがリダイレクトされるコールバック URL を入力します。
 - e. [次へ] を選択します。
 - f. [Continue] (続行) を選択し、[Register] (登録) を選択します。
13. 左のナビゲーションペインで [Keys (キー)] を選択します。
14. [Keys (キー)] ページで、[+] アイコンを選択します。
15. [Register a New Key] ページで、次の操作を行います。
 - a. [Key Name] に、キー名を入力します。
 - b. [Sign In with Apple] (Apple でサインイン) を選択後、[Configure] (設定) を選択します。
 - c. [Configure Key (キーの設定)] ページで、[Primary App ID (プライマリアプリ ID)]、[Save (保存)] の順に選択します。
 - d. [Continue] (続行) を選択し、[Register] (登録) を選択します。

Note

Apple でサインインをネイティブ iOS アプリケーションに統合するには、「[Apple でサインインしてユーザー認証を実装する](#)」を参照してください。

ネイティブ iOS 以外のプラットフォームで「Apple でサインイン」を統合する方法については、「[Apple JS でサインイン](#)」を参照してください。

Amazon Cognito フェデレーテッド ID コンソールで外部プロバイダーを設定する

以下の手順を使用して、外部プロバイダーを設定します。

Apple ID プロバイダー (IdP) を使ってサインインを追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [Apple でサインイン] を選択します。
5. [Apple Developer](#) で作成した OAuth プロジェクトのサービス ID を入力します。詳細については、Apple でサインインのドキュメントの「[Apple でサインインを使用してユーザーを認証する](#)」を参照してください。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。

- c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。

8. [変更を保存] を選択します。

Amazon Cognito フェデレーテッド ID で Apple をプロバイダーとして Apple でサインインするための CLI 例

この例では、IdP として Apple でサインインする MyIdentityPool という名前の ID プールを作成します。

```
aws cognito-identity create-identity-pool --identity-pool-name MyIdentityPool --supported-login-providers appleid.apple.com="sameple.apple.clientid"
```

詳細については、「[ID プールの作成](#)」を参照してください。

Amazon Cognito アイデンティティ ID を生成する

この例では、Amazon Cognito ID を生成 (または取得) します。これはパブリック API なので、この API を呼び出すために認証情報は必要ありません。

```
aws cognito-identity get-id --identity-pool-id SampleIdentityPoolId --logins appleid.apple.com="SignInWithAppleIdToken"
```

詳細については、「[get-id](#)」を参照してください。

Amazon Cognito アイデンティティ ID の認証情報を取得する

この例では、指定された ID および「Apple でサインイン」のログインの認証情報を返します。これはパブリック API なので、この API を呼び出すために認証情報は必要ありません。

```
aws cognito-identity get-credentials-for-identity --identity-id SampleIdentityId --logins appleid.apple.com="SignInWithAppleIdToken"
```

詳細については、「[get-credentials-for-identity](#)」を参照してください。

Apple でサインインを使用する: Android

Apple では、Android 版の「Apple でサインイン」をサポートする SDK を提供していません。代わりにウェブビューでウェブフローを使用できます。

- アプリケーションで「Apple でサインイン」を設定するには、Apple のドキュメントの「[「Apple でサインイン」するためのウェブページの設定](#)」を参照してください。
- Android ユーザーインターフェイスに [Apple でサインイン] ボタンを追加するには、Apple ドキュメントの「[ウェブ上に \[Appleでサインイン\] ボタンを表示する](#)」に従ってください。
- 「Apple でサインイン」を使用してユーザーを安全に認証するには、Apple のドキュメントの「[Apple でサインインするためのウェブページの設定](#)」を参照してください。

「Apple でサインイン」は、その状態の追跡にセッションオブジェクトを使用します。Amazon Cognito は、このセッションオブジェクトの ID トークンを使用してユーザーを認証し、一意の識別子を生成し、必要に応じてユーザーに他の AWS リソースへのアクセスを許可します。

```
@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);
}
```

「Apple でサインインを使用する」: iOS-Objective-C

Apple は、ネイティブ iOS アプリケーションでの「Apple でサインイン」の SDK サポートを提供しました。ネイティブ iOS デバイスで「Apple でサインイン」を使用してユーザー認証を実装するには、Apple のドキュメントで「[「Apple でサインイン」を使用したユーザー認証の実装](#)」を参照してください。

Amazon Cognito は ID トークンを使用してユーザーを認証し、一意の識別子を生成し、必要に応じてユーザーに他の AWS リソースへのアクセスを許可します。

```
(void)finishedWithAuth: (ASAuthorizationAppleIDCredential *)auth error: (NSError *)
error {
    NSString *idToken = [ASAuthorizationAppleIDCredential
objectForKey:@"identityToken"];
    credentialsProvider.logins = @{ "appleid.apple.com": idToken };
}
```

「Apple でサインイン」: iOS - Swift

Apple は、ネイティブ iOS アプリケーションでの「Apple でサインイン」の SDK サポートを提供しました。ネイティブ iOS デバイスで「Apple でサインイン」を使用してユーザー認証を実装するには、Apple のドキュメントで「[「Apple でサインイン」を使用したユーザー認証の実装](#)」を参照してください。

Amazon Cognito は ID トークンを使用してユーザーを認証し、一意の識別子を生成し、必要に応じてユーザーに他の AWS リソースへのアクセスを許可します。

iOS で「Apple でサインイン」を設定する方法についての詳細は、「[Apple でサインインを設定する](#)」を参照してください。

```
func finishedWithAuth(auth: ASAuthorizationAppleIDCredential!, error: NSError!) {
    if error != nil {
        print(error.localizedDescription)
    }
    else {
        let idToken = auth.identityToken,
            credentialsProvider.logins = ["appleid.apple.com": idToken!]
    }
}
```

「Apple でサインイン」を使用する: JavaScript

Apple では、JavaScript 用の「Apple でサインイン」をサポートする SDK は提供していません。代わりにウェブビューでウェブフローを使用できます。

- アプリケーションで「Apple でサインイン」を設定するには、Apple のドキュメントの「[「Apple でサインイン」するためのウェブページの設定](#)」を参照してください。
- JavaScript ユーザーインターフェイスに [Apple でサインイン] ボタンを追加するには、Apple ドキュメントの「[ウェブ上に \[Apple でサインイン\] ボタンを表示する](#)」に従ってください。
- 「Apple でサインイン」を使用してユーザーを安全に認証するには、Apple のドキュメントの「[Apple でサインインするためのウェブページの設定](#)」を参照してください。

「Apple でサインイン」は、その状態の追跡にセッションオブジェクトを使用します。Amazon Cognito は、このセッションオブジェクトの ID トークンを使用してユーザーを認証し、一意の識別子を生成し、必要に応じてユーザーに他の AWS リソースへのアクセスを許可します。

```
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.
    });
}
```

OIDC プロバイダーをアイデンティティプール IdP として設定する

[OpenID Connect](#) は認証のためのオープン標準で、多数のログインプロバイダーでサポートされています。Amazon Cognito を使用すると、[AWS Identity and Access Management](#) を使用して設定した OpenID Connect プロバイダーと ID をリンクできます。

OpenID Connect プロバイダーの追加

OpenID Connect プロバイダーの作成方法については、AWS Identity and Access Management ユーザーガイドの「[OpenID Connect \(OIDC\) ID プロバイダーの作成](#)」を参照してください。

Amazon Cognito とのプロバイダーの関連付け

OIDC ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. [OpenID Connect (OIDC)] を選択します。
5. AWS アカウント 内の IAM IdP から OIDC ID プロバイダーを選択します。新しい SAML プロバイダーを追加する場合は、[新しいプロバイダーの作成] を選択して IAM コンソールに移動します。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。

- その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
- a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
8. [変更を保存] を選択します。

複数の OpenID Connect プロバイダーを、単一の ID プールと関連付けることができます。

OpenID Connect の使用

サインインして ID トークンを受け取る方法については、プロバイダーのドキュメントを参照してください。

トークンを取得したら、ログインマップにトークンを追加します。プロバイダーの URI をキーとして使用します。

OpenID Connect トークンの検証

初めて Amazon Cognito と統合するときは、InvalidToken 例外を受け取る場合があります。Amazon Cognito が OpenID Connect (OIDC) トークンを検証する方法を理解しておくことが重要です。

Note

<https://tools.ietf.org/html/rfc7523> で規定されているように、Amazon Cognito では、システム間のクロックスキューを処理するために 5 分の猶予期間が提供されます。

1. iss パラメータは、ログインマップで使われるキー (login.provider.com など) と一致する必要があります。
2. 署名が有効である必要があります。署名は、RSA パブリックキーを通じて検証可能である必要があります。

Note

アイデンティティプールは、OIDC IdP 署名キーのキャッシュを短期間保持します。プロバイダーが署名キーを変更すると、このキャッシュが更新されるまで、Amazon Cognito が NoKeyFound エラーを返す可能性があります。このエラーが発生した場合は、アイデンティティプールが署名キーを更新するまで約 10 分待ちます。

3. 証明書の公開キーのフィンガープリントは、OIDC プロバイダーを作成したときに IAM で設定したフィンガープリントと一致します。
4. azp パラメータが存在する場合は、OIDC プロバイダーでリストされたクライアント ID に対してこの値を確認します。
5. azp パラメータが存在しない場合は、OIDC プロバイダーでリストされたクライアント ID に対して aud パラメータを確認します。

ウェブサイトの jwt.io は、これらの値を検証するためのトークンをデコードするための貴重なリソースです。

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 }
```

JavaScript

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: 'IDENTITY_POOL_ID',
  Logins: {
    'login.provider.com': token
  }
});
```

SAML プロバイダーをアイデンティティプール IdP として設定する

Amazon Cognito アイデンティティプールを使用すると、SAML 2.0 を介して ID プロバイダー (IdP) を使用してユーザーを認証できます。SAML をサポートする IdP を Amazon Cognito で使用して、ユーザーにシンプルなオンボーディングフローを提供することができます。SAML サポート IdP は、ユーザーが引き受けることができる IAM ロールを指定します。このようにして、さまざまなユーザーがさまざまな権限のセットを受け取ることができます。

SAML プロバイダーの ID プールの設定

次のステップでは、SAML ベースの IdP が使用されるように ID プールを設定する方法について説明します。

Note

SAML プロバイダーをサポートするように ID プールを設定する前に、まず [IAM コンソール](#) で SAML ID プロバイダーを設定します。詳細については、IAM ユーザーガイドの「[サードパーティーの SAML ソリューションプロバイダーと AWS の統合](#)」を参照してください。

SAML ID プロバイダー (IdP) を追加するには

1. [Amazon Cognito コンソール](#) で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. SAML を選択します。

5. AWS アカウント 内の IAM IdP から SAML ID プロバイダーを選択します。新しい SAML プロバイダーを追加する場合は、[新しいプロバイダの作成] を選択して IAM コンソールに移動します。
6. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときにリクエストするロールを設定するには、[ロール設定] を設定します。
 - その IdP のユーザーに、認証済みロールを設定したときに設定したデフォルトロールを割り当てることも、ルール付きのロールを選択することもできます。
 - i. [ルールを使用してロールを選択] を選択した場合は、ユーザー認証のソースクレーム、クレームを比較する演算子、選択したロールと一致させる値、およびロールの割り当てが一致したときに割り当てるロールを入力します。別の条件に基づいて追加のルールを作成するには、[別のものを追加] を選択します。
 - ii. [ロールの解決] を選択します。ユーザーのクレームがルールに合わない場合は、認証情報を拒否するか、認証済みロールの認証情報を発行できます。
7. Amazon Cognito がこのプロバイダーで認証されたユーザーに認証情報を発行するときに割り当てるプリンシパルタグを変更するには、[アクセスコントロールの属性] を設定します。
 - a. プリンシパルタグを適用しない場合は、[非アクティブ] を選択します。
 - b. sub および aud クレームに基づいてプリンシパルタグを適用するには、[デフォルトマッピングを使用] を選択します。
 - c. プリンシパルタグへの属性の独自のカスタムスキーマを作成するには、[カスタムマッピングを使用] を選択します。次に、タグに表示したい各クレームから取得するタグキーを入力します。
8. [変更を保存] を選択します。

SAML IdP の設定

SAML プロバイダーを作成したら、SAML ID プロバイダーを設定し、IdP と AWS の間に証明書利用者の信頼を追加します。多くの IdP では、IdP が XML ドキュメントから証明書利用者情報と証明書を読み取るために使用できる URL を指定できます。には AWS、<https://signin.aws.amazon.com/static/saml-metadata.xml> を使用できます。次のステップでは、IdP からの SAML アサーションレスポンスを設定して、AWS に必要なクレームを入力します。クレーム設定の詳細については、「[認証レスポンスの SAML アサーションを設定する](#)」を参照してください。

SAML IdP の SAML メタデータに複数の署名証明書が含まれている場合、サインイン時に、SAML アサーションが SAML メタデータ内のいずれかの証明書と一致すると、SAML アサーションは有効であるとアイデンティティプールは判断します。

SAML を使用したユーザーロールのカスタマイズ

Amazon Cognito ID で SAML を使用する場合、エンドユーザーのロールをカスタマイズできます。Amazon Cognito は、SAML ベースの IdP で [拡張フロー](#)のみをサポートします。SAML ベースの IdP を使用するために、ID プールに認証されたロールや認証されていないロールを指定する必要はありません。https://aws.amazon.com/SAML/Attributes/Role クレーム属性は、カンマ区切りのロールおよびプロバイダー ARN のペアを 1 つ以上指定します。これらは、ユーザーが引き受けることができるロールです。IdP から入手できるユーザー属性情報に基づいてロール属性を設定するように SAML IdP を設定できます。SAML アサーションで複数のロールを受け取る場合は、getCredentialsForIdentity を呼び出すときにオプションの customRoleArn パラメータを設定します。ロールが SAML アサーションのクレームのロールと一致する場合、ユーザーはこの customRoleArn を想定します。

SAML IdP を使用したユーザーの認証

SAML ベースの IdP とフェデレーションするには、ユーザーが login. AWS federation を開始する URL を決定します。IdP AD FS 2.0 では、URL の形式は https://<fqdn>/adfs/ls/IdpInitiatedSignOn.aspx?loginToRp=urn:amazon:webservices です。

Amazon Cognito に SAML IdP のサポートを追加するには、まず iOS アプリケーションまたは Android アプリケーションから SAML ID プロバイダーを使用してユーザーを認証する必要があります。SAML IdP との統合と認証に使用するコードは、SAML プロバイダーに固有です。ユーザーが認証されたら、Amazon Cognito API を使用して、結果として得られた SAML アサーションを Amazon Cognito アイデンティティに提供できます。

アイデンティティプールの API リクエストの Logins マップで SAML アサーションを繰り返したり、再生したりすることはできません。SAML アサーションを再生すると、そのアサーション ID は以前の API リクエストの ID と重複します。Logins マップ内の SAML アサーションを受け入れることができる API オペレーションとしては、[GetId](#)、[GetCredentialsForIdentity](#)、[GetOpenIdToken](#)、[GetOpenIdTokenForDeveloperIdentity](#) があります。SAML アサーション ID は、ID プール認証フローの API リクエストごとに 1 回再生できます。例えば、同じ SAML アサーションを GetId リクエストと後続の GetCredentialsForIdentity リクエストでは指定できますが、2 番目の GetId リクエストでは指定できません。

デベロッパーが認証した ID

Amazon Cognito は、[アイデンティティプール IdP として Facebook を設定する](#)、[アイデンティティプール IdP として Google を設定する](#)、[アイデンティティプール IdP として Login with Amazon を設定する](#)、および [アイデンティティプール IdP として Apple でサインインを設定する](#) 経由でのウェブ ID フェデレーションに加えて、デベロッパーが認証したアイデンティティもサポートします。開発者が認証した ID を使用すると、Amazon Cognito を使用してユーザーデータを同期し、AWS リソースにアクセスしながら、独自の既存の認証プロセスを通じてユーザーを登録および認証できます。デベロッパーが認証したアイデンティティの使用には、エンドユーザーのデバイス、認証のバックエンド、および Amazon Cognito 間の対話に関連します。詳細については、AWS ブログの「[Understanding Amazon Cognito Authentication Part 2: Developer Authenticated Identities](#)」を参照してください。

認証のフローについて

[GetOpenIdTokenForDeveloperIdentity](#) API オペレーションは、拡張認証と基本認証の両方についてデベロッパー認証を開始できます。この API は、管理者認証情報を使用してリクエストを認証します。Logins マップは、カスタム識別子とペアになっている login.mydevprovider などの、アイデンティティプールデベロッパープロバイダー名です。

例:

```
"Logins": {
  "login.mydevprovider": "my developer identifier"
}
```

拡張認証

[GetCredentialsForIdentity](#) API オペレーションを、GetOpenIdTokenForDeveloperIdentity からトークンの名前 cognito-identity.amazonaws.com と値を含む Logins マップで呼び出します。

例:

```
"Logins": {
  "cognito-identity.amazonaws.com": "eyJra12345EXAMPLE"
}
```

デベロッパー認証 ID を持つ GetCredentialsForIdentity は、アイデンティティプールのデフォルト認証ロールの一時的な認証情報を返します。

基本認証

[AssumeRoleWithWebIdentity](#) API オペレーションを呼び出し、適切な[信頼関係が定義されている](#) IAM ロールの RoleArn をリクエストします。WebIdentityToken の値を GetOpenIdTokenForDeveloperIdentity から取得したトークンに設定します。

デベロッパーが認証した ID の認証フローの詳細と、外部プロバイダー ID との違いについては、「[ID プールの認証フロー](#)」を参照してください。

デベロッパー名を指定して ID プールに関連付ける

デベロッパーが認証したアイデンティティを使用するには、デベロッパープロバイダーに関連付けられたアイデンティティプールが必要です。そのためには、以下の手順を実行します。

カスタムデベロッパープロバイダーを追加するには

1. [Amazon Cognito コンソール](#)で [ID プールの管理] をクリックします。アイデンティティプールを選択します。
2. [ユーザーアクセス] タブを選択します。
3. [ID プロバイダーを追加] を選択します。
4. カスタムデベロッパープロバイダーを選択します。
5. デベロッパープロバイダー名を入力します。デベロッパープロバイダーを追加した後は、変更または削除することはできません。
6. [変更を保存] を選択します。

注意: プロバイダー名を設定した場合、それを変更することはできません。

ID プロバイダーの実装

Android

デベロッパーが認証した ID を使用するには、AWSAbstractCognitoIdentityProvider を拡張する独自の ID プロバイダークラスを実装します。ID プロバイダークラスは、属性としてトークンを含むレスポンスオブジェクトを返す必要があります。

以下に示しているのは、ID プロバイダーの基本的な例です。

```
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;
    }
}
}
}

```

この ID プロバイダーを使用するには、これを `CognitoCachingCredentialsProvider` に渡す必要があります。例を示します。

```

DeveloperAuthenticationProvider developerProvider = new
    DeveloperAuthenticationProvider( null, "IDENTITYPOOLID", context, Regions.USEAST1);
CognitoCachingCredentialsProvider credentialsProvider = new
    CognitoCachingCredentialsProvider( context, developerProvider, Regions.USEAST1);

```

iOS - Objective-C

デベロッパーが認証した ID を使用するには、[AWSCognitoCredentialsProviderHelper](#) を拡張する独自の ID プロバイダークラスを実装します。ID プロバイダークラスは、属性としてトークンを含むレスポンスオブジェクトを返す必要があります。

```

@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

```

この ID プロバイダーを使用するには、次の例に示すように、それを `AWSCognitoCredentialsProvider` に渡します。

```

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];

```

認証されていない ID と開発者が認証した ID の両方をサポートする場合は、`AWSCognitoCredentialsProviderHelper` の実装で `logins` メソッドをオーバーライドします。

```

- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else{
        return [super logins];
    }
}

```

開発者が認証した ID とソーシャルプロバイダーをサポートする場合は、`AWSCognitoCredentialsProviderHelper` の `logins` 実装で現在のプロバイダーが誰であることを管理する必要があります。

```

- (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

デベロッパーが認証した ID を使用するには、[AWSCognitoCredentialsProviderHelper](#) を拡張する独自の ID プロバイダークラスを実装します。ID プロバイダークラスは、属性としてトークンを含むレスポンスオブジェクトを返す必要があります。

```

import AWSCore
/*
 * Use the token method to communicate with your backend to get an
 * identityId and token.
 */
class DeveloperAuthenticatedIdentityProvider : AWSCognitoCredentialsProviderHelper {
    override func token() -> AWSTask<NSString> {
        //Write code to call your backend:
        //pass username/password to backend or some sort of token to authenticate user, if
        successful,
        //from backend call getOpenIdTokenForDeveloperIdentity with logins map containing
        "your.provider.name":"enduser.username"
        //return the identity id and token to client
        //You can use AWSTaskCompletionSource to do this asynchronously

        // Set the identity id and return the token
        self.identityId = resultFromAbove.identityId
        return AWSTask(result: resultFromAbove.token)
    }
}

```

この ID プロバイダーを使用するには、次の例に示すように、それを `AWSCognitoCredentialsProvider` に渡します。

```

let devAuth =
DeveloperAuthenticatedIdentityProvider(regionType: .YOUR_IDENTITY_POOL_REGION,
identityPoolId: "YOUR_IDENTITY_POOL_ID", useEnhancedFlow: true,
identityProviderManager:nil)

```

```
let credentialsProvider =
    AWSCognitoCredentialsProvider(regionType: .YOUR_IDENTITY_POOL_REGION,
    identityProvider:devAuth)
let configuration = AWSServiceConfiguration(region: .YOUR_IDENTITY_POOL_REGION,
    credentialsProvider:credentialsProvider)
AWSServiceManager.default().defaultServiceConfiguration = configuration
```

認証されていない ID と開発者が認証した ID の両方をサポートする場合は、`AWSCognitoCredentialsProviderHelper` の実装で `logins` メソッドをオーバーライドします。

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else {
        return super.logins()
    }
}
```

開発者が認証した ID とソーシャルプロバイダーをサポートする場合は、`AWSCognitoCredentialsProviderHelper` の `logins` 実装で現在のプロバイダーが誰であるかを管理する必要があります。

```
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

バックエンドからアイデンティティ ID とセッショントークンを取得したら、それらを `AWS.CognitoIdentityCredentials` プロバイダーに渡します。以下に例を示します。

```
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

開発者が認証した ID を使用するには、CognitoAWSCredentials を拡張し、RefreshIdentity メソッドを上書きして、ユーザー ID とトークンをバックエンドから取得して返す必要があります。「example.com」という仮定のバックグラウンドに接続する ID プロバイダーの簡単な例を以下に示します。

```
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);
}
}

```

上記のコードでは、スレッドディスパッチャーオブジェクトを使用してコルーチンを呼び出します。プロジェクトでこれを行う方法がない場合は、シーンで以下のスクリプトを使用できます。

```

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

開発者が認証した ID を使用するには、CognitoAWSCredentials を拡張し、RefreshIdentity メソッドを上書きして、ユーザー ID とトークンをバックエンドから取得して返す必要があります。「example.com」という仮定のバックグラウンドに接続する ID プロバイダーの基本的な例を以下に示します。

```
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;
    }
}
```

ログインマップの更新 (Android および iOS のみ)

Android

ご使用の認証システムでユーザーが正常に認証されたら、デベロッパープロバイダー名と開発ユーザー ID を使用してログインマップを更新します。これは、認証システム内のユーザーを一意に識別する英数字の文字列です。refresh が変更されている可能性があるため、ログインマップを更新した後は、必ず `identityId` メソッドを呼び出してください。

```
HashMap<String, String> loginsMap = new HashMap<String, String>();
loginsMap.put(developerAuthenticationProvider.getProviderName(),
    developerUserIdentifier);

credentialsProvider.setLogins(loginsMap);
credentialsProvider.refresh();
```

iOS - Objective-C

認証情報がないか、または期限切れ場合、iOS SDK は `logins` メソッドを呼び出して最新のログインマップを取得するのみです。SDK で新しい認証情報の入手を強制する場合は (たとえば、エンドユーザーが認証されていない状態から認証され、認証されたユーザーに対して認証情報を必要とするなど)、`clearCredentials` で `credentialsProvider` を呼び出します。

```
[credentialsProvider clearCredentials];
```

iOS - Swift

認証情報がないか、または期限切れ場合、iOS SDK は `logins` メソッドを呼び出して最新のログインマップを取得するのみです。SDK で新しい認証情報の入手を強制する場合は (たとえば、エンドユーザーが認証されていない状態から認証され、認証されたユーザーに対して認証情報を必要とするなど)、`clearCredentials` で `credentialsProvider` を呼び出します。

```
credentialsProvider.clearCredentials()
```

トークンの取得 (サーバー)

[GetOpenIdTokenForDeveloperIdentity](#) を呼び出してトークンを取得します。この API は、AWS 開発者認証情報を使用してバックエンドから呼び出す必要があります。クライアント SDK から呼び

出すことはできません。API は Cognito アイデンティティプール ID (ID プロバイダー名をキーとして、識別子を値として含むログインマップ) と、オプションで Cognito アイデンティティ ID (認証されていないユーザーを認証する場合など) を受け取ります。識別子はユーザーのユーザー名、メールアドレス、または数値場合があります。API は、ユーザーの一意の Cognito ID と、エンドユーザーの OpenID Connect トークンを使用して呼び出しに応答します。

GetOpenIdTokenForDeveloperIdentity によって返されるトークンについては、以下の点に注意してください。

- トークンのカスタムの有効期限を指定して、トークンをキャッシュすることができます。カスタムの有効期限を指定しない場合、トークンは 15 分間有効です。
- 設定できる最大のトークンの期間は 24 時間です。
- トークンの期間を増やすことによるセキュリティへの影響について注意してください。攻撃者がこのトークンを取得した場合、そのトークンをエンドユーザーの AWS 認証情報とトークン期間交換できます。

以下の Java スニペットは、Amazon Cognito クライアントを初期化して、デベロッパーが認証したアイデンティティのトークンを取得する方法を示しています。

```
// 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 * 151);
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
//...
```

前述の手順に従って、アプリでデベロッパーが認証したアイデンティティを統合できます。問題またはご質問がある場合は、お気軽に[フォーラム](#)に投稿してください。

既存のソーシャル ID への接続

デベロッパーが認証したアイデンティティを使用する際のプロバイダーのすべてのリンク作成は、バックエンドから行う必要があります。ユーザーのソーシャル ID (Login with Amazon、「Apple でサインイン」、Facebook、または Google) にカスタム ID を接続するには、[GetOpenIdTokenForDeveloperIdentity](#) を呼び出すときに、ログインマップに ID プロバイダーのトークンを追加します。これを可能にするため、クライアント SDK からバックエンドを呼び出してエンドユーザーを認証するときは、エンドユーザーのソーシャルプロバイダートークンを追加で渡します。

たとえば、カスタム ID の Facebook へのリンクを試みている場合、`GetOpenIdTokenForDeveloperIdentity` を呼び出すときに、ID プロバイダー ID に加えて Facebook のトークンをログインマップに追加します。

```
logins.put("YOUR_IDENTITY_PROVIDER_NAME", "YOUR_END_USER_IDENTIFIER");
logins.put("graph.facebook.com", "END_USERS_FACEBOOK_ACCESSTOKEN");
```

プロバイダー間の移行のサポート

Android

アプリケーションでは、デベロッパーが認証したアイデンティティとともに、パブリックプロバイダー (Login with Amazon、「Apple でサインイン」、Facebook、または Google) を使用した、認証

されていないアイデンティティまたは認証されたアイデンティティのサポートが必要になる場合があります。デベロッパーが認証したアイデンティティとその他のアイデンティティ (認証されていないアイデンティティと、パブリックプロバイダーを使用して認証されたアイデンティティ) の重要な違いは、identityId およびトークンの取得方法にあります。他のアイデンティティについては、モバイルアプリケーションが認証システムに接続する代わりに Amazon Cognito と直接やり取りします。そのため、モバイルアプリケーションは、アプリユーザーが行った選択に基づいて、2 つの明確の流れをサポートする必要があります。そのためには、カスタム ID プロバイダーにいくつか変更を加える必要があります。

refresh メソッドはログインマップをチェックします。マップが空でなくデベロッパープロバイダー名を含むキーがある場合は、バックエンドを呼び出します。それ以外の場合は、getIdentityId メソッドを呼び出して NULL を返します。

```
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;
    }
}
```

同様に、getIdentityId メソッドにも、ログインマップの内容に応じて 2 つのフローがあります。

```
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

アプリケーションでは、デベロッパーが認証したアイデンティティとともに、パブリックプロバイダー (Login with Amazon、「Apple でサインイン」、Facebook、または Google) を使用した、認証されていないアイデンティティまたは認証されたアイデンティティのサポートが必要になる場合があります。これを行うには、[AWSCognitoCredentialsProviderHelper](#) logins メソッドをオーバーライドし、現在の ID プロバイダーに基づいて正しいログインマップを返せるようにします。この例では、非認証、Facebook 認証、デベロッパー認証を切り替える方法を示しています。

```
- (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];
}
}
```

非認証から認証に移行するときは、`[credentialsProvider clearCredentials];` を呼び出して、認証された新しい認証情報の取得を SDK に強制します。2 つの認証されたプロバイダーを切り替え、2 つのプロバイダーへのリンクを試みない (ログインディクショナリで複数のプロバイダー用のトークンを提供していない) 場合は、`[credentialsProvider clearKeychain];` を呼び出します。これにより、認証情報とアイデンティティの両方がクリアされ、SDK は新しい認証を取得します。

iOS - Swift

アプリケーションでは、デベロッパーが認証したアイデンティティとともに、パブリックプロバイダー (Login with Amazon、「Apple でサインイン」、Facebook、または Google) を使用した、認証されていないアイデンティティまたは認証されたアイデンティティのサポートが必要になる場合があります。これを行うには、[AWSCognitoCredentialsProviderHelper](#) `logins` メソッドをオーバーライドし、現在の ID プロバイダーに基づいて正しいログインマップを返せるようにします。この例では、非認証、Facebook 認証、デベロッパー認証を切り替える方法を示しています。

```
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()
    }
}
```


非認証から認証に移行するときは、`credentialsProvider.clearCredentials()` を呼び出して、認証された新しい認証情報の取得を SDK に強制します。2 つの認証されたプロバイダーを切り替え、2 つのプロバイダーへのリンクを試みない (ログインディクショナリで複数のプロバイダー用のトークンを提供していない) 場合は、`credentialsProvider.clearKeychain()` を呼び出します。これにより、認証情報とアイデンティティの両方がクリアされ、SDK は新しい認証を取得します。

Unity

アプリケーションでは、デベロッパーが認証したアイデンティティとともに、パブリックプロバイダー (Login with Amazon、「Apple でサインイン」、Facebook、または Google) を使用した、認証されていないアイデンティティまたは認証されたアイデンティティのサポートが必要になる場合があります。デベロッパーが認証したアイデンティティとその他のアイデンティティ (認証されていないアイデンティティと、パブリックプロバイダーを使用して認証されたアイデンティティ) の重要な違いは、`identityId` およびトークンの取得方法にあります。他のアイデンティティについては、モバイルアプリケーションが認証システムに接続する代わりに Amazon Cognito と直接やり取りします。モバイルアプリケーションは、アプリユーザーが行った選択に基づいて、2 つの明確の流れをサポートする必要があります。そのためには、カスタム ID プロバイダーにいくつか変更を加える必要があります。

Unity でこれを行うための推奨の方法は、`AbstractCognitoIdentityProvider` の代わりに `AmazonCognitoEnhancedIdentityProvider` から ID プロバイダーを拡張し、独自のメソッドの代わりに親 `RefreshAsync` メソッドを呼び出すことです (ユーザーが独自のバックエンドで認証されない場合)。ユーザーが認証されている場合、前に説明したのと同じ流れを使用できます。

Xamarin

アプリケーションでは、デベロッパーが認証したアイデンティティとともに、パブリックプロバイダー (Login with Amazon、「Apple でサインイン」、Facebook、または Google) を使用した、認証されていないアイデンティティまたは認証されたアイデンティティのサポートが必要になる場合があります。デベロッパーが認証したアイデンティティとその他のアイデンティティ (認証されていないアイデンティティと、パブリックプロバイダーを使用して認証されたアイデンティティ) の重要な違いは、`identityId` およびトークンの取得方法にあります。他のアイデンティティについては、モバイルアプリケーションが認証システムに接続する代わりに Amazon Cognito と直接やり取りします。モバイルアプリケーションは、アプリユーザーが行った選択に基づいて、2 つの明確の流れをサポートする必要があります。そのためには、カスタム ID プロバイダーにいくつか変更を加える必要があります。

認証されていないユーザーから認証されたユーザーへの切り替え

Amazon Cognito ID プールは、認証されたユーザーと認証されていないユーザーの両方をサポートします。認証されていないユーザーは、ID プロバイダー (IdPs) でログインしていない場合でも、AWS リソースにアクセスできます。このレベルのアクセスは、ユーザーがログインする前にユーザーにコンテンツを表示する場合に便利です。認証されていない各ユーザーは、個別にログインして認証していない場合でも、ID プールに一意の ID があります。

このセクションでは、ユーザーがログインに使用する ID を、認証されていない ID から認証された ID に切り替える方法について説明します。

Android

ユーザーは、認証されていないゲストとしてアプリケーションにログインできます。最終的に、ユーザーはサポートされている IdP のいずれかを使用してログインすることを決定する可能性があります。Amazon Cognito は、古いアイデンティティが新しいアイデンティティと同じ識別子を維持し、プロフィールデータが自動的にマージされることを確実にします。

アプリケーションに、プロフィールの結合が `IdentityChangedListener` インターフェイスを通じて通知されます。インターフェイスで `identityChanged` メソッドを実装して、これらのメッセージを受け取ります。

```
@override
public void identityChanged(String oldIdentityId, String newIdentityId) {
    // handle the change
}
```

iOS - Objective-C

ユーザーは、認証されていないゲストとしてアプリケーションにログインできます。最終的に、ユーザーはサポートされている IdP のいずれかを使用してログインすることを決定する可能性があります。Amazon Cognito は、古いアイデンティティが新しいアイデンティティと同じ識別子を維持し、プロフィールデータが自動的にマージされることを確実にします。

`NSNotificationCenter` はプロフィールの結合をアプリケーションに通知します。

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(identityIdDidChange:)
                                         name:AWSCognitoIdentityIdChangedNotification
```

```
        object:nil]);

-(void)identityDidChange:(NSNotification*)notification {
    NSDictionary *userInfo = notification.userInfo;
    NSLog(@"identity changed from %@ to %@",
          [userInfo objectForKey:AWSCognitoNotificationPreviousId],
          [userInfo objectForKey:AWSCognitoNotificationNewId]);
}
```

iOS - Swift

ユーザーは、認証されていないゲストとしてアプリケーションにログインできます。最終的に、ユーザーはサポートされている IdP のいずれかを使用してログインすることを決定する可能性があります。Amazon Cognito は、古いアイデンティティが新しいアイデンティティと同じ識別子を維持し、プロファイルデータが自動的にマージされることを確実にします。

NSNotificationCenter はプロファイルの結合をアプリケーションに通知します。

```
[NSNotificationCenter defaultCenter().addObserver(observer: self
    selector:"identityDidChange"
    name:AWSCognitoIdentityIdChangedNotification
    object:nil)

func identityDidChange(notification: NSNotification!) {
    if let userInfo = notification.userInfo as? [String: AnyObject] {
        print("identity changed from: \(userInfo[AWSCognitoNotificationPreviousId])
            to: \(userInfo[AWSCognitoNotificationNewId])")
    }
}
```

JavaScript

認証されていないユーザーとしての開始

通常、ユーザーは認証されていないロールから開始します。このロールでは、Logins プロパティを使用しないで設定オブジェクトの認証情報プロパティを設定します。この場合、デフォルト設定は次のようになります。

```
// set the default config object
var creds = new AWS.CognitoIdentityCredentials({
```

```
IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030'  
});  
AWS.config.credentials = creds;
```

認証されたユーザーへの切り替え

認証されていないユーザーが IdP にログインしたときに、トークンがあれば、カスタム関数を呼び出して認証情報オブジェクトを更新し Logins トークンを追加することで、認証されていないユーザーを認証されたユーザーに切り替えることができます。

```
// 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;  
}
```

また、CognitoIdentityCredentials オブジェクトを作成することもできます。作成する場合は、既存のサービスオブジェクトの認証情報プロパティをリセットして、更新された認証情報の設定内容を反映する必要があります。「[グローバル設定オブジェクトの使用](#)」を参照してください。

CognitoIdentityCredentials オブジェクトの詳細については、AWS SDK for JavaScript API リファレンスの[AWS「.CognitoIdentityCredentials」](#)を参照してください。

Unity

ユーザーは、認証されていないゲストとしてアプリケーションにログインできます。最終的に、ユーザーはサポートされている IdP のいずれかを使用してログインすることを決定する可能性があります。Amazon Cognito は、古いアイデンティティが新しいアイデンティティと同じ識別子を維持し、プロフィールデータが自動的にマージされることを確実にします。

プロフィールの結合の通知を受け取るには、IdentityChangedEvent にサブスクライブできます。

```
credentialsProvider.IdentityChangedEvent += delegate(object sender,  
    CognitoAWSCredentials.IdentityChangedEventArgs e)  
{
```

```
// handle the change
Debug.log("Identity changed from " + e.OldIdentityId + " to " + e.NewIdentityId);
};
```

Xamarin

ユーザーは、認証されていないゲストとしてアプリケーションにログインできます。最終的に、ユーザーはサポートされている IdP のいずれかを使用してログインすることを決定する可能性があります。Amazon Cognito は、古い ID の一意の識別子を新しい ID に継承して、プロファイルデータが自動的に結合されるようにします。

```
credentialsProvider.IdentityChangedEvent += delegate(object sender,
    CognitoAWSCredentials.IdentityChangedArgs e){
    // handle the change
    Console.WriteLine("Identity changed from " + e.OldIdentityId + " to " +
        e.NewIdentityId);
};
```

Amazon Cognito Sync

⚠ Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。

このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito Sync は、デバイス間でアプリケーション関連のユーザーデータを同期できるようにする AWS のサービス および クライアントライブラリです。Amazon Cognito Sync は、独自のバックエンドを使用せずに、モバイルデバイスとウェブ間でユーザープロフィールデータを同期できます。クライアントライブラリはローカルにデータをキャッシュするため、アプリはデバイスの接続状態にかかわらず、データを読み書きすることができます。デバイスがオンラインになると、データを同期できます。プッシュ同期を設定すると、更新が利用できることが他のデバイスにすぐに通知されます。

Amazon Cognito ID を使用できるリージョンの詳細については、「[AWS リージョン別のサービス](#)」を参照してください。

Amazon Cognito Sync の詳細については、以下のトピックを参照してください。

トピック

- [Amazon Cognito Sync の使用開始方法](#)
- [クライアント間でのデータの同期](#)
- [イベントコールバックの処理](#)
- [プッシュ同期の実装](#)
- [Amazon Cognito Sync ストリームの実装](#)
- [Amazon Cognito イベントによるワークフローのカスタマイズ](#)

Amazon Cognito Sync の使用開始方法

⚠ Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。

このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito Sync は、アプリケーション関連のユーザーデータのデバイス間同期を可能にする AWS サービスおよびクライアントライブラリです。これは、モバイルデバイスとウェブの全体でユーザープロファイルデータを同期させるために使用できます。クライアントライブラリはローカルにデータをキャッシュするため、アプリはデバイスの接続状態にかかわらず、データを読み書きすることができます。デバイスがオンラインのときは、データを同期し、プッシュ同期を設定すると、更新が利用できることが他のデバイスにすぐに通知されます。

Amazon Cognito で ID プールをセットアップする

Amazon Cognito Sync では、ユーザー ID を提供するための Amazon Cognito ID プールが必要です。最初にアイデンティティプールをセットアップしてから Amazon Cognito Sync を使用する必要があります。ID プールを作成し、SDK をインストールするには、[Amazon Cognito アイデンティティプールの開始方法](#) を参照してください。

データを保存し、同期する

ID プールを設定して SDK をインストールしたら、デバイス間でデータを保存して同期を開始できます。詳細については、「[クライアント間でのデータの同期](#)」を参照してください。

クライアント間でのデータの同期

⚠ Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。

このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito では、キーバリューペアが含まれるデータセットにユーザーデータを保存することができます。Amazon Cognito は、このデータを ID プールの ID と関連付け、アプリがログインやデバイス全体でアクセスできるようにします。このデータを Amazon Cognito サービスとエンドユーザーのデバイス間で同期するには、同期メソッドを呼び出します。各データセットは、最大 1 MB のサイズにすることができます。ID には、最大 20 個のデータセットを関連付けることができます。

Amazon Cognito Sync クライアントは、ID データ用のローカルキャッシュを作成します。アプリは、キーを読み取るか書き込むときに、このローカルキャッシュとやり取りします。このやり取りにより、オフラインの状態でも、デバイス上で行われたすべての変更がすぐに利用できるようになりますことが保証されます。同期メソッドが呼び出された場合、サービスからの変更はデバイスにプルされ、ローカルの変更はサービスにプッシュされます。この時点で、他のデバイスが変更を利用して同期できるようになります。

Amazon Cognito Sync クライアントの初期化

Amazon Cognito Sync クライアントを初期化するには、まず認証情報プロバイダーを作成する必要があります。認証情報プロバイダーは一時的な AWS 認証情報を取得して、アプリが AWS リソースにアクセスできるようにします。また、必要なヘッダーファイルをインポートする必要があります。以下のステップを使用して、Amazon Cognito Sync クライアントを初期化します。

Android

1. 「[認証情報の取得](#)」の指示に従って、認証情報プロバイダーを作成します。
2. Amazon Cognito パッケージを以下のようにインポートします: `import com.amazonaws.mobileconnectors.cognito.*;`
3. Amazon Cognito Sync を初期化 以下のように、Android アプリのコンテキスト、ID プールの ID、AWS リージョン、および初期化された Amazon Cognito の認証情報プロバイダーを渡します。

```
CognitoSyncManager client = new CognitoSyncManager(  
    getApplicationContext(),  
    Regions.YOUR_REGION,  
    credentialsProvider);
```


iOS - Objective-C

1. 「[認証情報の取得](#)」の指示に従って、認証情報プロバイダーを作成します。
2. 以下のように、AWSCore および Cognito をインポートし、AWSCognito を初期化します。

```
#import <AWSiOSSDKv2/AWSCore.h>
#import <AWSCognitoSync/Cognito.h>

AWSCognito *syncClient = [AWSCognito defaultCognito];
```

3. CocoaPods を使用している場合は、<AWSiOSSDKv2/AWSCore.h> を AWSCore.h と置き換えてください。Amazon Cognito のインポートにも同じ構文に従います。

iOS - Swift

1. 「[認証情報の取得](#)」の指示に従って、認証情報プロバイダーを作成します。
2. 以下のように AWSCognito をインポートし、初期化します。

```
import AWSCognito
let syncClient = AWSCognito.default()!
```

JavaScript

1. [Amazon Cognito Sync Manager for JavaScript](#) をダウンロードします。
2. プロジェクトに Sync Manager ライブラリを含めます。
3. 「[認証情報の取得](#)」の指示に従って、認証情報プロバイダーを作成します。
4. Sync Manager を以下のように初期化します。

```
var syncManager = new AWS.CognitoSyncManager();
```

Unity

1. 「[認証情報の取得](#)」の指示に従って、CognitoAWSCredentials のインスタンスを作成します。

2. CognitoSyncManager のインスタンスを作成します。以下のよう

に、CognitoAwsCredentials オブジェクトと AmazonCognitoSyncConfig を渡し、少なくともリージョン・セットを含めます。

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =  
    REGION };  
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Xamarin

1. 「[認証情報の取得](#)」の指示に従って、CognitoAWSCredentials のインスタンスを作成します。

2. CognitoSyncManager のインスタンスを作成します。以下のよう

に、CognitoAwsCredentials オブジェクトと AmazonCognitoSyncConfig を渡し、少なくともリージョン・セットを含めます。

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =  
    REGION };  
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

データセットについて

Amazon Cognito は、ユーザーのプロファイルデータをデータセットに編成します。各データセットはキーバリューペアの形式で、最大 1 MB のデータを含むことができます。データセットは、同期することができる最も細かいエンティティです。データセットで実行される読み取りと書き込みのオペレーションは、同期メソッドが呼び出されるまでしか、ローカルストアに影響しません。Amazon Cognito は一意の文字列によってデータセットを識別します。以下のように新しいデータセットを作成するか、既存のデータセットを開くことができます。

Android

```
Dataset dataset = client.openOrCreateDataset("datasetname");
```

データセットを削除するには、以下のように、最初にそれをローカルストレージから削除するためのメソッドを呼び出し、次に synchronize メソッドを呼び出して Amazon Cognito からそのデータセットを削除します。

```
dataset.delete();  
dataset.synchronize(syncCallback);
```

iOS - Objective-C

```
AWSCognitoDataset *dataset = [syncClient openOrCreateDataset:@"myDataSet"];
```

データセットを削除するには、以下のように、最初にそれをローカルストレージから削除するためのメソッドを呼び出し、次に `synchronize` メソッドを呼び出して Amazon Cognito からそのデータセットを削除します。

```
[dataset clear];  
[dataset synchronize];
```

iOS - Swift

```
let dataset = syncClient.openOrCreateDataset("myDataSet")!
```

データセットを削除するには、最初にそれをローカルストレージから削除するためのメソッドを呼び出し、次に `synchronize` メソッドを呼び出して Amazon Cognito からそのデータセットを削除します。

```
dataset.clear()  
dataset.synchronize()
```

JavaScript

```
syncManager.openOrCreateDataset('myDatasetName', function(err, dataset) {  
    // ...  
});
```

Unity

```
string myValue = dataset.Get("myKey");  
dataset.Put("myKey", "newValue");
```

データセットからキーを削除するには、以下のように `Remove` を設定します。

```
dataset.Remove("myKey");
```

Xamarin

```
Dataset dataset = syncManager.OpenOrCreateDataset("myDatasetName");
```

データセットを削除するには、以下のように、最初にそれをローカルストレージから削除するためのメソッドを呼び出し、次に `synchronize` メソッドを呼び出して Amazon Cognito からそのデータセットを削除します。

```
dataset.Delete();  
dataset.SynchronizeAsync();
```

データセットのデータの読み取りと書き込み

Amazon Cognito データセットはディクショナリとして機能し、値はキーでアクセスできます。データセットのキーと値は、データセットがディクショナリであるかのように読み取り、追加、または変更できます。

データセットに書き込む値は、同期メソッドを呼び出すまで、データのローカルにキャッシュされたコピーのみに影響することに注意してください。

Android

```
String value = dataset.get("myKey");  
dataset.put("myKey", "my value");
```

iOS - Objective-C

```
[dataset setString:@"my value" forKey:@"myKey"];  
NSString *value = [dataset stringForKey:@"myKey"];
```

iOS - Swift

```
dataset.setString("my value", forKey:"myKey")  
let value = dataset.stringForKey("myKey")
```

JavaScript

```
dataset.get('myKey', function(err, value) {
    console.log('myRecord: ' + value);
});

dataset.put('newKey', 'newValue', function(err, record) {
    console.log(record);
});

dataset.remove('oldKey', function(err, record) {
    console.log(success);
});
```

Unity

```
string myValue = dataset.Get("myKey");
dataset.Put("myKey", "newValue");
```

Xamarin

```
//obtain a value
string myValue = dataset.Get("myKey");

// Create a record in a dataset and synchronize with the server
dataset.OnSyncSuccess += SyncSuccessCallback;
dataset.Put("myKey", "myValue");
dataset.SynchronizeAsync();

void SyncSuccessCallback(object sender, SyncSuccessEventArgs e) {
    // Your handler code here
}
```

Android

以下のように、remove メソッドを使用して、データセットからキーを削除できます。

```
dataset.remove("myKey");
```

iOS - Objective-C

データセットからキーを削除するには、以下のように `removeObjectForKey` を設定します。

```
[dataset removeObjectForKey:@"myKey"];
```

iOS - Swift

データセットからキーを削除するには、以下のように `removeObjectForKey` を設定します。

```
dataset.removeObjectForKey("myKey")
```

Unity

データセットからキーを削除するには、以下のように `Remove` を設定します。

```
dataset.Remove("myKey");
```

Xamarin

`Remove` を使用して、データセットからキーを削除できます。

```
dataset.Remove("myKey");
```

同期ストアでのローカルデータの同期

Android

`synchronize` メソッドは、ローカルキャッシュデータと Amazon Cognito Sync ストアに格納されたデータを比較します。リモート変更は Amazon Cognito Sync ストアからプルされ、競合が発生した場合は競合解決が呼び出されます。そして、デバイス上の更新された値がサービスにプッシュされます。データセットを同期するには、その `synchronize` メソッドを呼び出します。

```
dataset.synchronize(syncCallback);
```

`synchronize` メソッドは、以下に説明する `SyncCallback` インターフェイスの実装を受け取ります。

`synchronizeOnConnectivity()` メソッドは、接続が利用可能な場合に同期を試みます。接続が直ちに利用可能な場合、`synchronizeOnConnectivity()` は `synchronize()` のように動作します。それ以外の場合、接続の変更をモニタリングし、接続が利用可能になると同期を実行します。`synchronizeOnConnectivity()` が複数回呼び出された場合、最後の同期リクエストのみが保持され、最後のコールバックのみが実行されます。データセットまたはコールバックがガーベージコレクションの対象になった場合、このメソッドは同期を実行せず、コールバックは実行されません。

データセットの同期と、さまざまなコールバックに関する詳細については、「[イベントコールバックの処理](#)」を参照してください。

iOS - Objective-C

`synchronize` メソッドは、ローカルキャッシュデータと Amazon Cognito Sync ストアに格納されたデータを比較します。リモート変更は Amazon Cognito Sync ストアからプルされ、競合が発生した場合は競合解決が呼び出されます。そして、デバイス上の更新された値がサービスにプッシュされます。データセットを同期するには、その `synchronize` メソッドを呼び出します。

`synchronize` メソッドは非同期で、応答を処理するために `AWSTask` オブジェクトを返します。

```
[[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;
}];
```

`synchronizeOnConnectivity` メソッドは、デバイスが接続されているときに同期を試みます。最初に、`synchronizeOnConnectivity` は接続を確認し、デバイスがオンラインである場合、すぐに `synchronize` を呼び出して、試行に関連付けられた `AWSTask` オブジェクトを返します。

デバイスがオフラインの場合、`synchronizeOnConnectivity` は、1) 次回にデバイスがオンラインになったときに同期をスケジュールし、2) 結果が `nil` の `AWSTask` を返します。スケジュールされた同期が有効なのは、データセットオブジェクトのライフサイクルの間のみです。接続が回復する前にアプリが終了された場合、データは同期されません。予定された同期中にイベントが発生した場合

に通知を受けるようにする場合は、AWS Cognito で見つかった通知のオブザーバーを追加する必要があります。

データセットの同期と、さまざまなコールバックに関する詳細については、「[イベントコールバックの処理](#)」を参照してください。

iOS - Swift

`synchronize` メソッドは、ローカルキャッシュデータと Amazon Cognito Sync ストアに格納されたデータを比較します。リモート変更は Amazon Cognito Sync ストアからプルされ、競合が発生した場合は競合解決が呼び出されます。そして、デバイス上の更新された値がサービスにプッシュされます。データセットを同期するには、その `synchronize` メソッドを呼び出します。

`synchronize` メソッドは非同期で、応答を処理するために `AWSTask` オブジェクトを返します。

```
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
})
```

`synchronizeOnConnectivity` メソッドは、デバイスが接続されているときに同期を試みます。最初に、`synchronizeOnConnectivity` は接続を確認し、デバイスがオンラインである場合、すぐに `synchronize` を呼び出して、試行に関連付けられた `AWSTask` オブジェクトを返します。

デバイスがオフラインの場合、`synchronizeOnConnectivity` は、1) 次回にデバイスがオンラインになったときに同期をスケジュールし、2) 結果が `nil` の `AWSTask` オブジェクトを返します。スケジュールされた同期が有効なのは、データセットオブジェクトのライフサイクルの間のみです。接続が回復する前にアプリが終了された場合、データは同期されません。予定された同期中にイベントが発生した場合に通知を受けるようにする場合は、AWS Cognito で見つかった通知のオブザーバーを追加する必要があります。

データセットの同期と、さまざまなコールバックに関する詳細については、「[イベントコールバックの処理](#)」を参照してください。

JavaScript

`synchronize` メソッドは、ローカルキャッシュデータと Amazon Cognito Sync ストアに格納されたデータを比較します。リモート変更は Amazon Cognito Sync ストアからプルされ、競合が発生した場合は競合解決が呼び出されます。そして、デバイス上の更新された値がサービスにプッシュされます。データセットを同期するには、その `synchronize` メソッドを呼び出します。

```
dataset.synchronize();
```

データセットの同期と、さまざまなコールバックに関する詳細については、「[イベントコールバックの処理](#)」を参照してください。

Unity

同期メソッドは、ローカルキャッシュデータと Amazon Cognito Sync ストアに格納されたデータを比較します。リモート変更は Amazon Cognito Sync ストアからプルされ、競合が発生した場合は競合解決が呼び出されます。そして、デバイス上の更新された値がサービスにプッシュされます。データセットを同期するには、その `synchronize` メソッドを呼び出します。

```
dataset.Synchronize();
```

同期は非同期で実行され、データセット内で指定できる複数のコールバックの 1 つを呼び出します。

データセットの同期と、さまざまなコールバックに関する詳細については、「[イベントコールバックの処理](#)」を参照してください。

Xamarin

`synchronize` メソッドは、ローカルキャッシュデータと Amazon Cognito Sync ストアに格納されたデータを比較します。リモート変更は Amazon Cognito Sync ストアからプルされ、競合が発生した場合は競合解決が呼び出されます。そして、デバイス上の更新された値がサービスにプッシュされます。データセットを同期するには、その `synchronize` メソッドを呼び出します。

```
dataset.SynchronizeAsync();
```

データセットの同期と、さまざまなコールバックに関する詳細については、「[イベントコールバックの処理](#)」を参照してください。

イベントコールバックの処理

 Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。

このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito Sync デベロッパーとして、さまざまなコールバックを実装して、さまざまな同期イベントやシナリオを処理できます。Android SDK の SyncCallback インターフェイスは、データセットが正常にダウンロードされたときの onSuccess()、例外が発生したときの onFailure()、ローカルデータとリモートデータの間の競合を解決するための onConflict() などの、データセットの同期に関する通知を設定します。

iOS SDK では、AWSCognitoDidStartSynchronizeNotification などの同様の通知に登録し、AWSCognitoRecordConflictHandler などのハンドラーを設定して競合を解決できます。JavaScript、Unity、Xamarin のプラットフォームには、同様のコールバックメカニズムがあります。これらのコールバックを実装すると、アプリケーションは Amazon Cognito Sync の使用時に発生する可能性のあるさまざまな同期イベントやシナリオを正常に処理できます。

Android

SyncCallback インターフェイス

SyncCallback インターフェイスを実装することで、データセットの同期に関するアプリの通知を受け取ることができます。これによりアプリは、ローカルデータの削除、認証されていないプロファイルや認証されたプロファイルの結合、同期の競合の解決に関するアクティブな決定ができます。インターフェイスで必要となる次のメソッドを実装する必要があります。

- onSuccess()
- onFailure()
- onConflict()
- onDatasetDeleted()
- onDatasetsMerged()

すべてのコールバックを指定しない場合は、すべてのコールバックに対して空のデフォルト実装を提供する、DefaultSyncCallback クラスを使用することもできます。

onSuccess

onSuccess() コールバックは、同期ストアから正常にデータセットをダウンロードしたときにトリガーされます。

```
@Override
public void onSuccess(Dataset dataset, List<Record> newRecords) {
}
```

onFailure

onFailure() は、同期中に例外が発生した場合に呼び出されます。

```
@Override
public void onFailure(DataStorageException dse) {
}
```

onConflict

ローカルストアと同期ストアで同じキーが変更された場合に、競合が発生する可能性があります。onConflict() メソッドは、競合の解決を処理します。このメソッドを実装しない場合、Amazon Cognito Sync クライアントはデフォルトで最新の変更を使用します。

```
@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

データセットが削除されると、Amazon Cognito クライアントは SyncCallback インターフェイスを使用して、データセットのローカルキャッシュコピーも削除されるべきかどうかを確認します。onDatasetDeleted() メソッドを実装して、クライアント SDK にローカルデータの処理方法を指定します。

```
@Override
public boolean onDatasetDeleted(Dataset dataset, String datasetName) {
    // return true to delete the local copy of the dataset
    return true;
}
```

onDatasetMerged

以前に接続されていない 2 つの ID がリンクされ、すべてのデータセットが結合されます。アプリケーションは、onDatasetsMerged() メソッドを通じて結合について通知されます。

```
@Override
public boolean onDatasetsMerged(Dataset dataset, List<String> datasetNames) {
    // return false to handle Dataset merge outside the synchronization callback
    return false;
}
```

iOS - Objective-C

同期の通知

Amazon Cognito クライアントは、同期コール中に多数の NSNotification イベントを生成します。標準の NSNotificationCenter を通じて、これらの通知をモニタリングするよう登録できます。

```
[NSNotificationCenter defaultCenter]
addObserver:self
```

```
selector:@selector(myNotificationHandler:)
name:NOTIFICATION_TYPE
object:nil];
```

Amazon Cognito は、以下にリストされている 5 つの通知タイプをサポートします。

AWSCognitoDidStartSynchronizeNotification

同期操作の開始中に呼び出されます。userInfo は、同期されているデータセットの名前であるキーデータセットを含みます。

AWSCognitoDidEndSynchronizeNotification

同期操作が完了した (正常またはその反対) ときに呼び出されます。userInfo は、同期されているデータセットの名前であるキーデータセットを含みます。

AWSCognitoDidFailToSynchronizeNotification

同期操作が失敗すると呼び出されます。userInfo は、同期されているデータセットの名前であるキーデータセットと、失敗の原因となったエラーを含むキーエラーを含みます。

AWSCognitoDidChangeRemoteValueNotification

ローカル変更が Amazon Cognito に正常にプッシュされたときに呼び出されます。userInfo は、同期されているデータセットの名前であるキーデータセットと、プッシュされたレコードキーの NSArray を含むキーを含みます。

AWSCognitoDidChangeLocalValueFromRemoteNotification

ローカルの値が同期操作が原因で変更されたときに呼び出されます。userInfo は、同期されているデータセットの名前であるキーデータセットと、変更されたレコードキーの NSArray を含むキーを含みます。

競合解決ハンドラ

同期操作中に、ローカルストアと同期ストアで同じキーが変更された場合、競合が発生する可能性があります。競合解決ハンドラを設定していない場合、Amazon Cognito はデフォルトで最新の更新を選択します。

AWSCognitoRecordConflictHandler を実装して割り当てることで、デフォルトの競合解決は変更できます。AWSCognitoConflict 入力パラメータには、ローカルにキャッシュされたデータ

と、同期ストアの競合するレコードの両方の `AWSCognitoRecord` オブジェクトが含まれています。`AWSCognitoConflict` を使用すると、ローカルレコード `[conflict resolveWithLocalRecord]`、リモートレコード `[conflict resolveWithRemoteRecord]`、またはまったく新しい値 `[conflict resolveWithValue:value]` との競合を解決できます。このメソッドから `nil` が返されると、同期は続行できなくなり、次回に同期処理が開始されると、競合が再び発生します。

クライアントレベルで競合解決ハンドラを設定できます。

```
client.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
    AWSCognitoConflict *conflict) {
    // always choose local changes
    return [conflict resolveWithLocalRecord];
};
```

または、データセットレベルで設定できます。

```
dataset.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
    AWSCognitoConflict *conflict) {
    // override and always choose remote changes
    return [conflict resolveWithRemoteRecord];
};
```

データセットが削除されたハンドラ

データセットが削除されると、Amazon Cognito クライアントは `AWSCognitoDatasetDeletedHandler` を使用して、データセットのローカルキャッシュコピーも削除されるべきかどうかを確認します。`AWSCognitoDatasetDeletedHandler` が実装されていない場合、ローカルデータは自動的に消去されます。消去する前にローカルデータのコピーを維持するか、ローカルデータを維持する場合は、`AWSCognitoDatasetDeletedHandler` を実装します。

クライアントレベルでデータセットが削除されたハンドラを設定することができます。

```
client.datasetDeletedHandler = ^BOOL (NSString *datasetName) {
    // make a backup of the data if you choose
    ...
    // delete the local data (default behavior)
    return YES;
};
```

または、データセットレベルで設定できます。

```
dataset.datasetDeletedHandler = ^BOOL (NSString *datasetName) {  
    // override default and keep the local data  
    return NO;  
};
```

データセットの結合ハンドラ

以前に接続されていない 2 つの ID がリンクされ、すべてのデータセットが結合されます。アプリケーションは、DatasetMergeHandler メソッドを通じて結合について通知されます。ハンドラは、ルートデータセットの名前と、ルートデータセットの結合としてマーキングされているデータセット名の配列を受け取ります。

DatasetMergeHandler が実装されていない場合、これらのデータセットは無視されますが、ID の最大 20 個の合計データセットで引き続き領域を占有します。

クライアントレベルでデータセット結合ハンドラをセットアップすることもできます。

```
client.datasetMergedHandler = ^(NSString *datasetName, NSArray *datasets) {  
    // Blindly delete the datasets  
    for (NSString *name in datasets) {  
        AWSCognitoDataset *merged = [[AWSCognito defaultCognito]  
openOrCreateDataset:name];  
        [merged clear];  
        [merged synchronize];  
    }  
};
```

または、データセットレベルで設定できます。

```
dataset.datasetMergedHandler = ^(NSString *datasetName, NSArray *datasets) {  
    // Blindly delete the datasets  
    for (NSString *name in datasets) {  
        AWSCognitoDataset *merged = [[AWSCognito defaultCognito]  
openOrCreateDataset:name];  
        // do something with the data if it differs from existing dataset  
        ...  
        // now delete it  
        [merged clear];  
        [merged synchronize];  
    }  
};
```

iOS - Swift

同期の通知

Amazon Cognito クライアントは、同期コール中に多数の `NSNotification` イベントを生成します。標準の `NSNotificationCenter` を通じて、これらの通知をモニタリングするよう登録できます。

```
NSNotificationCenter.defaultCenter().addObserver(observer: self,
    selector: "myNotificationHandler",
    name:NOTIFICATION_TYPE,
    object:nil)
```

Amazon Cognito は、以下にリストされている 5 つの通知タイプをサポートします。

`AWSCognitoDidStartSynchronizeNotification`

同期操作の開始中に呼び出されます。`userInfo` は、同期されているデータセットの名前であるキーデータセットを含みます。

`AWSCognitoDidEndSynchronizeNotification`

同期操作が完了した (正常またはその反対) ときに呼び出されます。`userInfo` は、同期されているデータセットの名前であるキーデータセットを含みます。

`AWSCognitoDidFailToSynchronizeNotification`

同期操作が失敗すると呼び出されます。`userInfo` は、同期されているデータセットの名前であるキーデータセットと、失敗の原因となったエラーを含むキーエラーを含みます。

`AWSCognitoDidChangeRemoteValueNotification`

ローカル変更が Amazon Cognito に正常にプッシュされたときに呼び出されます。`userInfo` は、同期されているデータセットの名前であるキーデータセットと、プッシュされたレコードキーの `NSArray` を含むキーを含みます。

`AWSCognitoDidChangeLocalValueFromRemoteNotification`

ローカルの値が同期操作が原因で変更されたときに呼び出されます。`userInfo` は、同期されているデータセットの名前であるキーデータセットと、変更されたレコードキーの `NSArray` を含むキーを含みます。

競合解決ハンドラ

同期操作中に、ローカルストアと同期ストアで同じキーが変更された場合、競合が発生する可能性があります。競合解決ハンドラを設定していない場合、Amazon Cognito はデフォルトで最新の更新を選択します。

`AWSCognitoRecordConflictHandler` を実装して割り当てることで、デフォルトの競合解決は変更できます。`AWSCognitoConflict` 入力パラメータには、ローカルにキャッシュされたデータと、同期ストアの競合するレコードの両方の `AWSCognitoRecord` オブジェクトが含まれています。`AWSCognitoConflict` を使用すると、ローカルレコード [`conflict.resolveWithLocalRecord`]、リモートレコード [`conflict.resolveWithRemoteRecord`]、またはまったく新しい値 [`conflict.resolveWithValue.value`] との競合を解決できます。このメソッドから `nil` が返されると、同期は続行できなくなり、次回に同期処理が開始されると、競合が再び発生します。

クライアントレベルで競合解決ハンドラを設定できます。

```
client.conflictHandler = {
    (datasetName: String?, conflict: AWSCognitoConflict?) ->
    AWSCognitoResolvedConflict? in
        return conflict.resolveWithLocalRecord()
}
```

または、データセットレベルで設定できます。

```
dataset.conflictHandler = {
    (datasetName: String?, conflict: AWSCognitoConflict?) ->
    AWSCognitoResolvedConflict? in
        return conflict.resolveWithLocalRecord()
}
```

データセットが削除されたハンドラ

データセットが削除されると、Amazon Cognito クライアントは `AWSCognitoDatasetDeletedHandler` を使用して、データセットのローカルキャッシュコピーも削除されるべきかどうかを確認します。`AWSCognitoDatasetDeletedHandler` が実装されていない場合、ローカルデータは自動的に消去されます。消去する前にローカルデータのコピーを維持するか、ローカルデータを維持する場合は、`AWSCognitoDatasetDeletedHandler` を実装します。

クライアントレベルでデータセットが削除されたハンドラを設定することができます。

```
client.datasetDeletedHandler = {
    (datasetName: String!) -> Bool in
    // make a backup of the data if you choose
    ...
    // delete the local data (default behaviour)
    return true
}
```

または、データセットレベルで設定できます。

```
dataset.datasetDeletedHandler = {
    (datasetName: String!) -> Bool in
    // make a backup of the data if you choose
    ...
    // delete the local data (default behaviour)
    return true
}
```

データセットの結合ハンドラ

以前に接続されていない 2 つの ID がリンクされ、すべてのデータセットが結合されます。アプリケーションは、DatasetMergeHandler メソッドを通じて結合について通知されます。ハンドラは、ルートデータセットの名前と、ルートデータセットの結合としてマーキングされているデータセット名の配列を受け取ります。

DatasetMergeHandler が実装されていない場合、これらのデータセットは無視されますが、ID の最大 20 個の合計データセットで引き続き領域を占有します。

クライアントレベルでデータセット結合ハンドラをセットアップすることもできます。

```
client.datasetMergedHandler = {
    (datasetName: String!, datasets: [AnyObject]!) -> Void in
    for nameObject in datasets {
        if let name = nameObject as? String {
            let merged = AWS.Cognito.defaultCognito().openOrCreateDataset(name)
            merged.clear()
            merged.synchronize()
        }
    }
}
```

または、データセットレベルで設定できます。

```
dataset.datasetMergedHandler = {
  (datasetName: String!, datasets: [AnyObject]!) -> Void in
  for nameObject in datasets {
    if let name = nameObject as? String {
      let merged = AWSCognito.defaultCognito().openOrCreateDataset(name)
      // do something with the data if it differs from existing dataset
      ...
      // now delete it
      merged.clear()
      merged.synchronize()
    }
  }
}
```

JavaScript

同期のコールバック

データセットで `synchronize()` を実行するときに、オプションで次の各状態を処理するコールバックを指定できます。

```
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()

`onSuccess()` コールバックは、同期ストアから正常にデータセットを更新したときにトリガーされます。コールバックを定義しない場合、同期はユーザーの操作なしに成功します。

```
onSuccess: function(dataset, newRecords) {  
  console.log('Successfully synchronized ' + newRecords.length + ' new records.');}
```

onFailure()

`onFailure()`は、同期中に例外が発生した場合に呼び出されます。コールバックを定義しない場合、同期はユーザーの操作なしに失敗します。

```
onFailure: function(err) {  
  console.log('Synchronization failed.');  console.log(err);  
}
```

onConflict()

ローカルストアと同期ストアで同じキーが変更された場合に、競合が発生する可能性があります。`onConflict()` メソッドは、競合の解決を処理します。このメソッドを実行しない場合、競合があると同期は破棄されます。

```
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());  
  
  }  
  
  callback(resolved);  
}
```

```
    // 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()

データセットが削除されると、Amazon Cognito クライアントは `onDatasetDeleted()` コールバックを使用して、データセットのローカルキャッシュコピーも削除されるべきかどうかを確認します。デフォルトでは、データセットは削除されません。

```
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()

以前に接続されていない 2 つの ID がリンクされ、すべてのデータセットが結合されます。アプリケーションは、`onDatasetsMerged()` コールバックを通じて結合について通知されます。

```
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

データセットを開くか作成した後は、別のコールバックを設定できます。これは、Synchronize メソッドを使用したときにトリガーされます。この方法により、コールバックを登録することができます。

```
dataset.OnSyncSuccess += this.HandleSyncSuccess;  
dataset.OnSyncFailure += this.HandleSyncFailure;  
dataset.OnSyncConflict = this.HandleSyncConflict;  
dataset.OnDatasetMerged = this.HandleDatasetMerged;  
dataset.OnDatasetDeleted = this.HandleDatasetDeleted;
```

SyncSuccess と SyncFailure は、+= を = の代わりに使用し、複数のコールバックをサブスクライブできるようにします。

OnSyncSuccess

OnSyncSuccess コールバックは、クラウドから正常にデータセットを更新したときにトリガーされます。コールバックを定義しない場合、同期はユーザーの操作なしに成功します。

```
private void HandleSyncSuccess(object sender, SyncSuccessEvent e)  
{  
    // Continue with your game flow, display the loaded data, etc.  
}
```

OnSyncFailure

OnSyncFailure は、同期中に例外が発生した場合に呼び出されます。コールバックを定義しない場合、同期はユーザーの操作なしに失敗します。

```
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

ローカルストアと同期ストアで同じキーが変更された場合に、競合が発生する可能性があります。OnSyncConflict コールバックは競合の解決を処理します。このメソッドを実行しない場合、競合があると同期は破棄されます。

```
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

データセットが削除されると、Amazon Cognito クライアントは OnDatasetDeleted コールバックを使用して、データセットのローカルキャッシュコピーも削除されるべきかどうかを確認します。デフォルトでは、データセットは削除されません。

```
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

以前に接続されていない 2 つの ID がリンクされ、すべてのデータセットが結合されます。アプリケーションは、OnDatasetsMerged コールバックを通じて結合について通知されます。

```
public bool HandleDatasetMerged(Dataset localDataset, List<string> mergedDatasetNames)
{
    foreach (string name in mergedDatasetNames)
    {
        Dataset mergedDataset = syncManager.OpenOrCreateDataset(name);
        //Lambda function to delete the dataset after fetching it
        EventHandler<SyncSuccessEvent> lambda;
        lambda = (object sender, SyncSuccessEvent e) => {
            ICollection<string> existingValues = localDataset.GetAll().Values;
            ICollection<string> newValues = mergedDataset.GetAll().Values;

            //Implement your merge logic here

            mergedDataset.Delete(); //Delete the dataset locally
            mergedDataset.OnSyncSuccess -= lambda; //We don't want this callback to be
            fired again
            mergedDataset.OnSyncSuccess += (object s2, SyncSuccessEvent e2) => {
                localDataset.Synchronize(); //Continue the sync operation that was
            interrupted by the merge
            };
            mergedDataset.Synchronize(); //Synchronize it as deleted, failing to do so
            will leave us in an inconsistent state
            };
            mergedDataset.OnSyncSuccess += lambda;
            mergedDataset.Synchronize(); //Asnchronously fetch the dataset
        }

        // returning true allows the Synchronize to continue and false stops it
        return false;
    }
}
```


Xamarin

データセットを開くか作成した後は、別のコールバックを設定できます。これは、Synchronize メソッドを使用したときにトリガーされます。この方法により、コールバックを登録することができます。

```
dataset.OnSyncSuccess += this.HandleSyncSuccess;  
dataset.OnSyncFailure += this.HandleSyncFailure;  
dataset.OnSyncConflict = this.HandleSyncConflict;  
dataset.OnDatasetMerged = this.HandleDatasetMerged;  
dataset.OnDatasetDeleted = this.HandleDatasetDeleted;
```

SyncSuccess と SyncFailure は、+= を = の代わりに使用し、複数のコールバックをサブスクライブできるようにします。

OnSyncSuccess

OnSyncSuccess コールバックは、クラウドから正常にデータセットを更新したときにトリガーされます。コールバックを定義しない場合、同期はユーザーの操作なしに成功します。

```
private void HandleSyncSuccess(object sender, SyncSuccessEventArgs e)  
{  
    // Continue with your game flow, display the loaded data, etc.  
}
```

OnSyncFailure

OnSyncFailure は、同期中に例外が発生した場合に呼び出されます。コールバックを定義しない場合、同期はユーザーの操作なしに失敗します。

```
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

ローカルストアと同期ストアで同じキーが変更された場合に、競合が発生する可能性があります。OnSyncConflict コールバックは競合の解決を処理します。このメソッドを実行しない場合、競合があると同期は破棄されます。

```
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

データセットが削除されると、Amazon Cognito クライアントは OnDatasetDeleted コールバックを使用して、データセットのローカルキャッシュコピーも削除されるべきかどうかを確認します。デフォルトでは、データセットは削除されません。

```
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

以前に接続されていない 2 つの ID がリンクされ、すべてのデータセットが結合されます。アプリケーションは、OnDatasetsMerged コールバックを通じて結合について通知されます。

```
public bool HandleDatasetMerged(Dataset localDataset, List<string> mergedDatasetNames)
{
    foreach (string name in mergedDatasetNames)
    {
        Dataset mergedDataset = syncManager.OpenOrCreateDataset(name);

        //Implement your merge logic here

        mergedDataset.OnSyncSuccess += lambda;
        mergedDataset.SynchronizeAsync(); //Asnchronously fetch the dataset
    }

    // returning true allows the Synchronize to continue and false stops it
    return false;
}
```

プッシュ同期の実装

 Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito は自動的にアイデンティティとデバイス間の関係を追跡します。プッシュ同期機能を使用すると、ID データが変更されたときに、特定の ID のすべてのインスタンスに確実に通知されます。プッシュ同期は、特定のアイデンティティの同期ストアデータが変更されるたびに、そのアイデンティティに関連付けられているすべてのデバイスが、この変更を伝えるサイレントプッシュ通知を受け取ることを確実にします。

Note

プッシュ同期は JavaScript、Unity、Xamarin ではサポートされません。

プッシュ同期を使用する前に、まずプッシュ同期用にアカウントをセットアップし、Amazon Cognito コンソールでプッシュ同期を有効にする必要があります。

Amazon Simple Notification Service (Amazon SNS) アプリケーションを作成する

[SNS デベロッパーガイド](#)の説明に従って、サポートされているプラットフォーム用の Amazon SNS アプリを作成し、設定します。

Amazon Cognito コンソールでプッシュ同期を有効にする

プッシュ同期は Amazon Cognito コンソールを使って有効にできます。[コンソールのホームページ](#)から、次の操作を行います。

1. プッシュ同期を有効にする ID プールの名前をクリックします。アイデンティティプールの [Dashboard] (ダッシュボード) ページが表示されます。
2. [ダッシュボード] ページの右上にある、[ID プールの管理] をクリックします。[フェデレーティッドアイデンティティ] ページが表示されます。
3. 下にスクロールし、同期をプッシュしますをクリックして展開します。
4. [Service role] (サービスロール) ドロップダウンメニューで、SNS 通知を送信するアクセス許可を Cognito に付与する IAM ロールを選択します。[Create role] (ロールの作成) をクリックして、[AWS IAM コンソール](#)で ID プールに関連付けられたロールを作成または変更します。
5. プラットフォームアプリケーションを選択し、[Save Changes] (変更の保存) をクリックします。
6. アプリケーションへの SNS アクセスの許可

AWS Identity and Access Management コンソールで、完全な Amazon SNS アクセスを持つように IAM ロールを設定するか、完全な Amazon SNS アクセスを持つ新しいロールを作成します。次のロール信頼ポリシーの例では、Amazon Cognito Sync に IAM ロールを引き受ける制限付きの機能を付与しています。Amazon Cognito Sync は、条件 `aws:SourceArn` の ID プールと条件 `aws:SourceAccount` のアカウントの両方に代わり行う場合にのみ、その役割を担うことができます。

JSON

```
{
  "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"
        }
      }
    }
  ]
}
```

IAM ロールの詳細については、「[ロール \(委任とフェデレーション\)](#)」を参照してください。

アプリケーションでのプッシュ同期の使用: Android

アプリケーションは、Google Play サービスをインポートする必要があります。[Android SDK マネージャー](#)から、最新バージョンの Google Play SDK をダウンロードできます。Android のドキュメントの「[Android Implementation](#)」に従ってアプリを登録し、GCM から登録 ID を受け取ります。登録 ID を取得したら、以下のスニペットにあるように、デバイスを Amazon Cognito に登録する必要があります。

```
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);
}
```

これで、デバイスをサブスクライブして、特定のデータセットから更新を受け取ることができます。

```
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);
    }
}
```

データセットからプッシュ通知の受け取りを中止するには、unsubscribe メソッドを呼び出します。CognitoSyncManager オブジェクトのすべてのデータセット (または特定のサブセット) にサブスクライブするには、subscribeAll() を使用します。

```
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);
    }
}
```

[Android BroadcastReceiver](#) オブジェクトの実装で、最新バージョンの変更されたデータセットを確認して、アプリをもう一度同期する必要があるかどうか判断できます。

```
@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() {
        // ...
    });
}
}
```

プッシュ通知のペイロードでは、次のキーを利用できます。

- **source:** cognito-sync。これは、通知間の差別化要素として機能します。
- **identityPoolId:** ID プールの ID。これは検証や追加の情報用に使用できますが、レシーバーの観点からは不可欠ではありません。
- **identityId:** プール内のアイデンティティ ID。
- **datasetName:** 更新されたデータセットの名前。これは `openOrCreateDataset` 呼び出しの目的で利用できます。
- **syncCount:** リモートデータセットの同期カウント。これは、ローカルデータセットが古くなっていないこと、および入力同期が新しいことを確認する方法として使用できます。

アプリケーションでプッシュ同期を使用する: iOS - Objective-C

アプリ用のデバイストークンを取得するには、Apple ドキュメントの「[Registering for Remote Notifications](#)」を参照してください。APN から `NSData` オブジェクトとしてデバイストークンを受け取ったら、以下のように、Sync クライアントの `registerDevice:` メソッドを使用して、デバイスを Amazon Cognito に登録する必要があります。

```
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;
}
];
```

デバッグモードでは、デバイスは APN サンドボックスで登録されます。リリースモードでは、APN で登録されます。特定のデータセットからアップデートを受けるには、subscribe メソッドを使用します。

```
[[[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;
}
];
```

データセットからプッシュ通知の受け取りを中止するには、unsubscribe メソッドを呼び出します。

```
[[[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;
}
];
```


AWSCognito オブジェクトのすべてのデータセットをサブスクライブするには、`subscribeAll` を呼び出します。

```
[[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;
}];
```

`subscribeAll` を呼び出す前に、少なくとも各データセットで 1 回同期を行い、データセットがサーバー上に存在するようにします。

プッシュ通知に応答するには、アプリの代理で `didReceiveRemoteNotification` メソッドを実装する必要があります。

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:
(NSDictionary *)userInfo
{
    [[NSNotificationCenter defaultCenter]
postNotificationName:@"CognitoPushNotification" object:userInfo];
}
```

通知ハンドラを使用して通知を投稿する場合は、データセットへのハンドルがある、アプリケーションの他の場所で通知に応答することができます。次のように通知にサブスクライブしているとします。

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(didReceivePushSync:)
name: :@"CognitoPushNotification" object:nil];
```

その場合は、次のように通知に応答できます。

```
- (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;
    }];
}
```

プッシュ通知のペイロードでは、次のキーを利用できます。

- source: cognito-sync。これは、通知間の差別化要素として機能します。
- identityPoolId: ID プールの ID。これは検証や追加の情報用に使用できますが、レシーバーの観点からは不可欠ではありません。
- identityId: プール内のアイデンティティ ID。
- datasetName: 更新されたデータセットの名前。これは openOrCreateDataset 呼び出しの目的で利用できます。
- syncCount: リモートデータセットの同期カウント。これは、ローカルデータセットが古くなっていないこと、および入力同期が新しいことを確認する方法として使用できます。

アプリケーションでプッシュ同期を使用する: iOS - Swift

アプリ用のデバイストークンを取得するには、Apple ドキュメントの「Registering for Remote Notifications」を参照してください。APN から NSData オブジェクトとしてデバイストークンを受け取ったら、以下の のように、Sync クライアントの registerDevice: メソッドを使用して、デバイスを Amazon Cognito に登録する必要があります。

```
let syncClient = AWSCognito.default()
syncClient.registerDevice(devToken).continueWith(block: { (task: AWSTask!) ->
AnyObject! in
    if (task.error != nil) {
        print("Unable to register device: " + task.error.localizedDescription)

    } else {
        print("Successfully registered device with id: \(task.result)")
    }
})
```

```
        return task
    })
```

デバッグモードでは、デバイスは APN サンドボックスで登録されます。リリースモードでは、APN で登録されます。特定のデータセットからアップデートを受けるには、subscribe メソッドを使用します。

```
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
})
```

データセットからプッシュ通知の受け取りを中止するには、unsubscribe メソッドを呼び出します。

```
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
})
```

AWSCognito オブジェクトのすべてのデータセットをサブスクライブするには、subscribeAll を呼び出します。

```
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
})
```

subscribeAll を呼び出す前に、少なくとも各データセットで 1 回同期を行い、データセットがサーバー上に存在するようにします。

プッシュ通知に応答するには、アプリの代理で didReceiveRemoteNotification メソッドを実装する必要があります。

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo:
[NSObject : AnyObject],
    fetchCompletionHandler completionHandler: (UIBackgroundFetchResult) -> Void) {

    NSNotificationCenter.defaultCenter().postNotificationName("CognitoPushNotification",
    object: userInfo)
}
```

通知ハンドラを使用して通知を投稿する場合は、データセットへのハンドルがある、アプリケーションの他の場所で通知に応答することができます。次のように通知にサブスクライブしているとします。

```
NSNotificationCenter.defaultCenter().addObserver(observer:self,
    selector:"didReceivePushSync:",
    name:"CognitoPushNotification",
    object:nil)
```

その場合は、次のように通知に応答できます。

```
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
    }
}
}
```

プッシュ通知のペイロードでは、次のキーを利用できます。

- `source: cognito-sync`。これは、通知間の差別化要素として機能します。
- `identityPoolId`: ID プールの ID。これは検証や追加の情報用に使用できますが、レシーバーの観点からは不可欠ではありません。
- `identityId`: プール内のアイデンティティ ID。
- `datasetName`: 更新されたデータセットの名前。これは `openOrCreateDataset` 呼び出しの目的で利用できます。
- `syncCount`: リモートデータセットの同期カウント。これは、ローカルデータセットが古くなっていないこと、および入力同期が新しいことを確認する方法として使用できます。

Amazon Cognito Sync ストリームの実装

 Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito ストリームは、Amazon Cognito に保存されているデータに対する制御と洞察をデベロッパーに提供します。これで、デベロッパーはデータが更新および同期されるたびにイベントを受け取るように Kinesis ストリームを設定できるようになります。Amazon Cognito は、所有する Kinesis ストリームに各データセットの変更をリアルタイムでプッシュできます。

Amazon Cognito ストリームを使用すると、すべての Sync データを Kinesis に移動することができます。その後、これらをさらなる分析のために Amazon Redshift などのデータウェアハウスツールにストリーミングできます。Kinesis の詳細については、「[Amazon Kinesis の使用開始方法](#)」を参照してください。

ストリームの設定

Amazon Cognito ストリームは、Amazon Cognito コンソールで設定できます。Amazon Cognito コンソールで Amazon Cognito ストリームを有効にするには、パブリッシュ先の Kinesis ストリームと、選択されたストリームにイベントを配置するためのアクセス許可を Amazon Cognito に付与する IAM ロールを選択する必要があります。

[コンソールのホームページ](#)から、次の操作を行います。

1. Amazon Cognito ストリームをセットアップする ID プールの名前をクリックします。アイデンティティプールの [Dashboard] (ダッシュボード) ページが表示されます。
2. [ダッシュボード] ページの右上にある、[ID プールの管理] をクリックします。[Manage Federated Identities] (フェデレーティッド ID の管理) ページが表示されます。
3. 下にスクロールし、[Cognito ストリーム] をクリックして展開します。
4. [ストリーム名] ドロップダウンメニューで、既存の Kinesis ストリームの名前を選択します。または、[ストリームの作成] をクリックしてストリームを作成し、ストリーム名とシャード数を入力します。シャードに関する情報、およびストリームに必要なシャード数の見積りに関するヘルプについては、[Kinesis デベロッパーガイド](#)を参照してください。
5. [Publish role] (パブリッシュロール) ドロップダウンメニューで、ストリームをパブリッシュするためアクセス権限を Amazon Cognito に付与する IAM ロールを選択します。[Create role] (ロールの作成) をクリックして、[AWS IAM コンソール](#)で ID プールに関連付けられたロールを作成または変更します。
6. [ストリームの状態] ドロップダウンメニューで、[有効] を選択してストリームの更新を有効にします。[Save Changes] (変更の保存) をクリックします。

Amazon Cognito ストリームが正常に設定されると、この ID プール内のデータセットに対する後続の更新がすべてストリームに送信されるようになります。

ストリームのコンテンツ

ストリームに送信される各レコードは、単一の同期を表します。ストリームに送信されるレコードの例を次に示します。

```
{
  "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
}
```

Kinesis の最大ペイロードサイズである 1 MB を超える更新については、Amazon Cognito にその更新の完全なコンテンツが格納されている事前署名済みの Amazon S3 の URL が含まれます。

Amazon Cognito ストリームを設定した後で Kinesis ストリームを削除する、または Amazon Cognito Sync がロールを引き受けられないようにロール信頼のアクセス許可を変更する場合、Amazon Cognito ストリームを無効にします。Kinesis ストリームを再作成するか、ロールを修正してから、ストリームを再有効化する必要があります。

一括パブリッシュ

Amazon Cognito ストリームが設定されると、ID プール内の既存のデータに対して一括パブリッシュ操作を実行できるようになります。一括パブリッシュ操作をコンソール経由で開始する、または API 経由で直接開始すると、Amazon Cognito が、更新を受信しているものと同じストリームに対してこのデータのパブリッシュを開始します。

Amazon Cognito は、一括パブリッシュ操作の使用時にストリームに送信されるデータの一意性を保証しません。更新については、更新としてだけでなく、一括パブリッシュの一部として同じものを受け取る場合があります。ストリームからレコードを処理する際は、このことを念頭に置いてください。

すべてのストリームを一括パブリッシュするには、「ストリームの設定」の手順 1~6 に従ってから、[Start bulk publish] (一括パブリッシュを開始) をクリックします。一括パブリッシュ操作は、常に 1 つの進行中操作に制限され、24 時間ごとに 1 つの正常な一括パブリッシュリクエストに制限されます。

Amazon Cognito イベントによるワークフローのカスタマイズ

 Amazon Cognito Sync を初めて使用する場合は、[AWS AppSync](#) を使用してください。Amazon Cognito Sync と同様に、AWS AppSync はデバイス間でアプリケーションデータを同期するためのサービスです。このサービスは、アプリの設定やゲームステートといったユーザーデータの同期化を可能にします。また、複数のユーザーが同期し、共有されたデータでリアルタイムにコラボレートできるようにすることで、これらの機能を拡張します。

Amazon Cognito Events では、Amazon Cognito の重要なイベントに応じて AWS Lambda 関数を実行できます。Amazon Cognito は、データセットが同期されるときに Sync Trigger イベントを生成します。Sync Trigger イベントを使用して、ユーザーがデータを更新するときにアクションを実行できます。この関数は、クラウド内に保存し、ユーザーの他のデバイスに同期する前にデータを評価して、オプションで操作できます。これは、ユーザーの他のデバイスに対して同期する前に、デバイスから受信するデータを検証したり、プレーヤーが新しいレベルに達したときに賞を与えるなど、受信データに基づいてデータセットの他の値を更新する場合に役立ちます。

以下のステップでは、Amazon Cognito のデータセットが同期されるたびに実行される Lambda 関数のセットアップ手順を説明します。

Note

Amazon Cognito イベントを使用するときは、Amazon Cognito ID から取得した認証情報しか使用できません。関連付けられた Lambda 関数があるが、AWS アカウント認証情報 (開発者認証情報) UpdateRecords を使用して を呼び出す場合、Lambda 関数は呼び出されません。

での関数の作成 AWS Lambda

Lambda を Amazon Cognito と統合するには、最初に Lambda で関数を作成する必要があります。これを行うには、以下の手順を実行します。

Amazon Cognito での Lambda 関数の選択

1. Lambda コンソールを開きます。

2. [Create a Lambda function] (Lambda 関数の作成) をクリックします。
3. [Select blueprint] (設計図の選択) 画面で、「cognito-sync-trigger」を検索して選択します。
4. [Configure event sources] (イベントソースの設定) 画面で、イベントソースのタイプを [Cognito Sync Trigger] に設定し、ID プールを選択します。[Next] (次へ) をクリックします。

Note

Amazon Cognito Sync トリガーをコンソール外で設定する場合、Amazon Cognito が関数を呼び出せるように、Lambda リソースベースの権限を追加する必要があります。このアクセス許可は、Lambda コンソール (「[リソースベースのポリシーの使用 AWS Lambda](#)」を参照) または Lambda [AddPermission](#) オペレーションを使用して追加できます。

Lambda リソースベースのポリシーの例

次の AWS Lambda リソースベースのポリシーは、Amazon Cognito に Lambda 関数を呼び出す制限付きの機能を付与します。Amazon Cognito は、条件 `aws:SourceArn` の ID プールと条件 `aws:SourceAccount` のアカウントに代わりに関数を呼び出すことができます。

JSON

```
{
  "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": "arn:aws:lambda:us-east-1:111122223333:function:MyFunction",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-east-1:abcdefg-1234-5678-910a-0e8443553f95"
        }
      }
    }
  ]
}
```

```
    }  
  }  
}  
}
```

5. [Configure function] (関数の設定) 画面で、関数の名前と説明を入力します。[Runtime] (ランタイム) を「Node.js」に設定したままにします。コードは、例では変更しないまま使用します。デフォルトの例では、同期中のデータに変更を加えません。コードは、Amazon Cognito Sync Trigger イベントが発生したことのみを記録します。[Handler] (ハンドラ) の名前は「index.handler」設定のままにしておきます。[Role] (ロール) については、AWS Lambdaにアクセスするためのコード許可を付与する IAM ロールを選択します。ロールを変更する方法については、「IAM コンソール」を参照してください。[Advanced settings] (詳細設定) は変更しないまま使用します。[Next] (次へ) をクリックします。
6. [Review] (確認) 画面で詳細を確認し、[Create function] (関数の作成) をクリックします。次のページに新しい Lambda 関数が表示されます。

Lambda で適切な関数を作成したところで、この関数を Amazon Cognito Sync Trigger イベントのハンドラとして選択する必要があります。以下のステップで、このプロセスについて説明します。

コンソールのホームページから、以下の操作を行います。

1. Amazon Cognito イベントをセットアップする ID プールの名前をクリックします。ID プールのダッシュボードページが表示されます。
2. ダッシュボードページの右上にある [Manage Federated Identities] (フェデレーティッド ID の管理) をクリックします。[Manage Federated Identities] (フェデレーティッド ID の管理) ページが表示されます。
3. スクロールダウンし、[Cognito Events] (Cognito イベント) をクリックして展開します。
4. [Sync Trigger] ドロップダウンメニューで、Sync イベントが発生したときにトリガーする Lambda 関数を選択します。
5. [Save Changes] (変更の保存) をクリックします。

これで、データセットが同期されるたびに Lambda 関数が実行されるようになります。次のセクションでは、同期中に関数のデータの読み取りと修正を行う方法について説明します。

Sync trigger 用の Lambda 関数の記述

Sync Trigger は、サービスプロバイダインタフェースが使用するプログラミングパラダイムに従います。Amazon Cognito は、Lambda 関数に対して以下の JSON 形式の入力を提供します。

```
{
  "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 では、関数の戻り値が入力と同じ形式であることが想定されています。

Sync Trigger イベント用の関数を記述するときは、次の点に注意してください。

- Amazon Cognito が、UpdateRecords の実行時に Lambda 関数を呼び出すと、関数は 5 秒以内に応答する必要があります。応答しない場合、Amazon Cognito Sync サービスが LambdaSocketTimeoutException 例外をスローします。このタイムアウト値を大きくすることはできません。
- LambdaThrottledException 例外を取得した場合は、同期オペレーションをもう一度実行して、レコードを更新します。
- Amazon Cognito は、データセットに存在するすべてのレコードを関数に提供します。
- アプリユーザーが更新するレコードには、op フィールドが replace として設定されています。削除されたレコードには、op フィールドが remove として設定されています。
- アプリユーザーがレコードを更新していない場合でも、任意のレコードを変更できます。
- datasetRecords を除くすべてのフィールドは読み取り専用です。それらを変更しないでください。これらのフィールドを変更すると、レコードを更新することはできません。

- レコードの値を変更するには、値を更新し、op を replace に設定します。
- レコードを削除するには、op を remove に設定するか、値を null に設定します。
- レコードを追加するには、datasetRecords 配列に新しいレコードを追加します。
- Amazon Cognito は、Amazon Cognito がレコードを更新するときに、レスポンス内の省略されたレコードを無視します。

サンプル Lambda 関数

以下の Lambda 関数のサンプルでは、データへのアクセス、変更、削除の方法を示しています。

```
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);
};
```

Amazon Cognito のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、AWS お客様とお客様の間の責任共有です。[責任共有モデル](#)ではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。AWS また、では、安全に使用できるサービスも提供しています。サードパーティーの監査者は、[AWS コンプライアンスプログラム](#)コンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。Amazon Cognito に適用されるコンプライアンスプログラムの詳細については、「[コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム](#)」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、Amazon Cognito の使用時に責任共有モデルを適用する方法を理解するために役立ち、セキュリティおよびコンプライアンス上の目的を満たすように Amazon Cognito を設定する方法を説明します。また、Amazon Cognito リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

内容

- [Amazon Cognito でのデータ保護](#)
- [Amazon Cognito 向けの Identity and access management](#)
- [Amazon Cognito でのログインとモニタリング](#)
- [インターフェイスエンドポイント \(AWS PrivateLink\) を使用して Amazon Cognito にアクセスする](#)
- [Amazon Cognito のコンプライアンス検証](#)
- [Amazon Cognito の耐障害性](#)
- [Amazon Cognito のインフラストラクチャセキュリティ](#)
- [Amazon Cognito ユーザープールの設定と脆弱性分析](#)
- [AWS Amazon Cognito の マネージドポリシー](#)

Amazon Cognito でのデータ保護

責任 AWS [共有モデル](#)、Amazon Cognito (Amazon Cognito) でのデータ保護に適用されます。このモデルで説明されているように、AWS はすべての AWS クラウドを実行するグローバルインフラストラクチャを保護する責任があります。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。このコンテンツには、使用する AWS サービスのセキュリティ設定および管理タスクが含まれます。データプライバシーの詳細については、「[データプライバシーのよくある質問](#)」を参照してください。

データ保護の目的で、AWS アカウントの認証情報を保護し、AWS Identity and Access Management (IAM) を使用して個々のユーザーアカウントを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、次の方法でデータを保護することをお勧めします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、サービス内のすべての AWS デフォルトのセキュリティコントロールを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これにより、Amazon S3 に保存される個人データの検出と保護が支援されます。

顧客のアカウント番号などの機密の識別情報は、[名前] フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、または SDK を使用して Amazon Cognito AWS CLI または他の AWS サービスを使用する場合も同様です。AWS SDKs Amazon Cognito、またはその他のサービスに入力されたデータは、いずれも診断ログへの包含のために取得される可能性があります。外部サーバーへの URL を指定するときは、そのサーバーへのリクエストを検証するための認証情報を URL に含めないでください。

データ暗号化

データ暗号化は、通常、保管時の暗号化と転送中の暗号化の 2 つのカテゴリに分類されます。

保管時の暗号化

Amazon Cognito 内のデータは、業界標準に従って保管時に暗号化されます。

Amazon Cognito は、[検索可能な暗号化](#)を使用したユーザー属性検索で個人を特定できる情報の機密性、完全性、可用性をサポートしています。これらのハッシュベースのメッセージ認証コード (HMAC) 関数は、ユーザープールデータセット向けにパフォーマンスが最適化されており、ユーザー属性のプレーンテキスト値と暗号化された値の間でマッピングされます。Amazon Cognito は、ユーザープールを暗号化する KMS キーを使用して HMAC 値を計算します。この保護は、次の属性に適用されます。

- sub
- E メール
- phone_number
- given_name
- family_name
- 名前
- username
- preferred_username
- cognito:user_status

転送時の暗号化

マネージドサービスである Amazon Cognito は、AWS グローバルネットワークセキュリティで保護されています。AWS セキュリティサービスと [ガインフラストラクチャ AWS](#) を保護する方法については、[AWS 「クラウドセキュリティ」](#) を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「[Infrastructure Protection](#)」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で Amazon Cognito にアクセスします。クライアントは次をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

Amazon Cognito ユーザープールおよび ID プールには、IAM 認証、非認証、およびトークン認証による API オペレーションがあります。認証されていない API オペレーションやトークン認証された API オペレーションは、お客様 (アプリケーションのエンドユーザー) による使用を目的としていま

す。認証されていない API オペレーションとトークン認証された API オペレーションは、保管中および転送中に暗号化されます。詳細については、「[認可モデル別にグループ化された API オペレーションのリスト](#)」を参照してください。

Note

Amazon Cognito は、コンテンツを内部で暗号化します。お客様提供のキーはサポートしていません。

Amazon Cognito 向けの Identity and access management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つです。IAM 管理者は、誰を認証するか (サインインさせるか) と、誰に Amazon Cognito リソースの使用を許可するか (アクセス許可を付与するか) を制御します。IAM は、追加料金なしで利用できる AWS のサービス です。

トピック

- [オーディエンス](#)
- [アイデンティティを使用した認証](#)
- [ポリシーを使用したアクセスの管理](#)
- [Amazon Cognito で IAM が機能する仕組み](#)
- [Amazon Cognito の ID ベースのポリシー例](#)
- [Amazon Cognito のアイデンティティとアクセスのトラブルシューティング](#)
- [Amazon Cognito のサービスリンクロールの使用](#)

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「[Amazon Cognito のアイデンティティとアクセスのトラブルシューティング](#)」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「[Amazon Cognito で IAM が機能する仕組み](#)」を参照)

- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「[Amazon Cognito の ID ベースのポリシー例](#)」を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーテッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「[AWS アカウントにサインインする方法](#)」を参照してください。

プログラムによるアクセスの場合、は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「[API リクエストに対するAWS 署名バージョン 4](#)」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント ルートユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「[ルートユーザー認証情報が必要なタスク](#)」を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーテッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[IAM アイデンティティセンターとは](#)」を参照してください。

IAM ユーザーとグループ

[IAM ユーザー](#)は、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「[ID プロバイダーとのフェデレーションを使用してアクセスする必要がある AWS](#)」を参照してください。

[IAM グループ](#)は、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「[IAM ユーザーに関するユースケース](#)」を参照してください。

IAM ロール

[IAM ロール](#)は、特定のアクセス許可を持つアイデンティティであり、一時的な認証情報を提供します。[ユーザーから IAM ロール \(コンソール\) に切り替えるか、または API オペレーションを呼び出すことで、ロールを引き受けることができます。](#) AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「[ロールを引き受けるための各種方法](#)」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイドの「[IAM でのクロスアカウントリソースアクセス](#)」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられている場合のアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「[JSON ポリシー概要](#)」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイドの [カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する](#) を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「[管理ポリシーとインラインポリシーのいずれかを選択する](#)」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、[プリンシパルを指定する](#)必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「[IAM エンティティのアクセス許可境界](#)」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「[サービスコントロールポリシー](#)」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「[リソースコントロールポリシー \(RCP\)](#)」を参照してください。

- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「[セッションポリシー](#)」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「[ポリシー評価ロジック](#)」を参照してください。

Amazon Cognito で IAM が機能する仕組み

IAM を使用して Amazon Cognito へのアクセスを管理する前に、Amazon Cognito で利用できる IAM の機能を確認します。

Amazon Cognito で使用できる IAM の機能

IAM 機能	Amazon Cognito サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	あり
ポリシー条件キー	あり
ACL	なし
ABAC (ポリシー内のタグ)	部分的
一時認証情報	あり
プリンシパル権限	いいえ
サービスロール	あり

IAM 機能	Amazon Cognito サポート
サービスリンクロール	はい

Amazon Cognito およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要については、IAM ユーザーガイドの[AWS「IAM と連携する のサービス」](#)を参照してください。

Amazon Cognito の ID ベースのポリシー

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「[カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する](#)」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「[IAM JSON ポリシーの要素のリファレンス](#)」を参照してください。

Amazon Cognito の ID ベースのポリシー例

Amazon Cognito の ID ベースのポリシー例については、「[Amazon Cognito の ID ベースのポリシー例](#)」を参照してください。

Amazon Cognito 内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、[プリンシパルを指定する](#)必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、またはを含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。詳細については、IAM ユーザーガイドの[IAM でのクロスアカウントリソースアクセス](#)を参照してください。

Amazon Cognito のポリシーアクション

ポリシーアクションのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの Action 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Amazon Cognito アクションのリストについては、「サービス認可リファレンス」の「[Amazon Cognito ID で定義されるアクション](#)」を参照してください。

Amazon Cognito のポリシーアクションでは、アクションの前に、次のプレフィックスを使用します。

```
cognito-identity
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [  
    "cognito-identity:action1",  
    "cognito-identity:action2"  
]
```

署名された API と署名されていない API

AWS 認証情報を使用して Amazon Cognito API リクエストに署名すると、AWS Identity and Access Management (IAM) ポリシーで制限できます。AWS 認証情報で署名する必要がある API リクエストには、サーバー側での AdminInitiateAuth を使用したサインイン、および UpdateUserPool などの Amazon Cognito リソースを作成、表示、または変更するアクションが含まれます。署名付き API リクエストの詳細については、[AWS 「API リクエストの署名」](#)を参照してください。

Amazon Cognito は、一般公開するアプリケーション用のコンシューマー ID 製品であるため、次の署名されていない API にアクセスできます。アプリケーションは、ユーザーと見込みユーザーに代

わってこれらの API リクエストを行います。一部の API (InitiateAuth など) は、新しい認証セッションを開始するために、事前の認可を必要としません。一部の API (VerifySoftwareToken など) は、アクセストークンやセッションキーを認可に使用して、既存の認証済みセッションを持つユーザーの MFA 設定を完了します。署名されていない、承認済みの Amazon Cognito ユーザープール API は、[Amazon Cognito API リファレンス](#)で示しているように、リクエスト構文での Session または AccessToken パラメータをサポートしています。署名されていない Amazon Cognito アイデンティティ API は、[Amazon Cognito フェデレーティッドアイデンティティ API リファレンス](#)で示しているように、IdentityId パラメータをサポートしています。

Amazon Cognito ユーザープール API オペレーションの認可モデルとロールの詳細については、「[認可モデル別にグループ化された API オペレーションのリスト](#)」を参照してください。

Amazon Cognito アイデンティティプール API オペレーション

- GetId
- GetOpenIdToken
- GetCredentialsForIdentity
- UnlinkIdentity

Amazon Cognito ユーザープール API オペレーション

- 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

Amazon Cognito の ID ベースのポリシー例については、「[Amazon Cognito の ID ベースのポリシー例](#)」を参照してください。

Amazon Cognito のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、[Amazon リソースネーム \(ARN\)](#) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ワイルドカード (*) を使用することで、ステートメントがすべてのリソースに適用されることを示します。

```
"Resource": "*"

```

Amazon リソースネーム (ARN)

Amazon Cognito フェデレーティッド ID の ARN

Amazon Cognito ID プール (フェデレーティッド ID) では、以下の例にあるように、Amazon リソースネーム (ARN) 形式を使用して IAM ユーザーのアクセスを特定の ID プールに制限することができます。ARN の詳細については、「[IAM 識別子](#)」を参照してください。

```
arn:aws:cognito-identity:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID
```

Amazon Cognito Sync の ARN

Amazon Cognito Sync では、カスタマーがアイデンティティプール ID、アイデンティティ ID、およびデータセット名でアクセスを制限することもできます。

ID プールで動作する API の場合、サービス名が `cognito-identity` ではなく `cognito-sync` である場合を除いて、ID プールの ARN 形式が Amazon Cognito フェデレーティッド ID のものと同じになります。

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID
```

`RegisterDevice` などの単一のアイデンティティで動作する API については、以下の ARN 形式で個々のアイデンティティを参照できます。

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID/  
identity/IDENTITY_ID
```

データセットで動作する API (例えば、`UpdateRecords` および `ListRecords`) の場合、次の ARN 形式を使用して個別のデータセットを参照できます。

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID/  
identity/IDENTITY_ID/dataset/DATASET_NAME
```

Amazon Cognito ユーザープールの ARN

Amazon Cognito ユーザープールでは、以下の ARN 形式を使用して、ユーザーのアクセスを特定のユーザープールに制限することが可能です。

```
arn:aws:cognito-idp:REGION:ACCOUNT_ID:userpool/USER_POOL_ID
```

Amazon Cognito リソースのタイプと ARN のリストについては、「サービス認可リファレンス」の「[Amazon Cognito ID で定義されるリソースタイプ](#)」を参照してください。各リソースの ARN を指定できるアクションについては、「[Amazon Cognito で定義されるアクション](#)」を参照してください。

Amazon Cognito の ID ベースのポリシー例については、「[Amazon Cognito の ID ベースのポリシー例](#)」を参照してください。

Amazon Cognito のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの[条件演算子](#)を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」の[AWS 「グローバル条件コンテキストキー」](#)を参照してください。

Amazon Cognito の条件キーのリストについては、「サービス認可リファレンス」の「[Amazon Cognito ID の条件キー](#)」を参照してください。条件キーを使用できるアクションとリソースについては、「[Amazon Cognito で定義されるアクション](#)」を参照してください。

Amazon Cognito の ID ベースのポリシー例については、「[Amazon Cognito の ID ベースのポリシー例](#)」を参照してください。

Amazon Cognito のアクセスコントロールリスト (ACL)

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon Cognito の属性ベースのアクセス制御 (ABAC)

ABAC (ポリシー内のタグ) のサポート: 一部

属性ベースのアクセスコントロール (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの[条件要素](#)でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「[ABAC 認可でアクセス許可を定義する](#)」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「[属性ベースのアクセスコントロール \(ABAC\) を使用する](#)」を参照してください。

Amazon Cognito での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、AWS リソースへの短期的なアクセスを提供し、フェデレーションまたはスウィッチロールの使用時に自動的に作成されます。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「IAM ユーザーガイド」の「[IAM の一時的なセキュリティ認証情報](#)」と「[IAM と連携するAWS のサービス](#)」を参照してください。

Amazon Cognito のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) のサポート: なし

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS リクエストを行う際のポリシーの詳細については、「[転送アクセスセッション](#)」を参照してください。

Amazon Cognito のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける [IAM ロール](#)です。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、IAM ユーザーガイドの [AWS のサービスに許可を委任するロールを作成する](#)を参照してください。

Amazon Cognito のサービスロールの詳細については、「[プッシュ同期を有効にする](#)」と「[プッシュ同期の実装](#)」を参照してください。

Warning

サービスロールのアクセス許可を変更すると、Amazon Cognito の機能が破損する可能性があります。Amazon Cognito が指示する場合以外は、サービスロールを編集しないでください。

Amazon Cognito のサービスにリンクされたロール

サービスリンクロールのサポート: あり

サービスにリンクされたロールは、にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

Amazon Cognito のサービスにリンクされたロールの作成または管理の詳細については、「[Amazon Cognito のサービスリンクロールの使用](#)」を参照してください。

Amazon Cognito の ID ベースのポリシー例

デフォルトでは、ユーザーおよびロールには Amazon Cognito リソースを作成または変更するアクセス許可がありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「[IAM ポリシーを作成する \(コンソール\)](#)」を参照してください。

Amazon Cognito が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「[Amazon Cognito のアクション、リソース、および条件キー](#)」を参照してください。

トピック

- [ポリシーに関するベストプラクティス](#)
- [Amazon Cognito コンソールの使用](#)

- [自分の権限の表示をユーザーに許可する](#)
- [特定の ID プールにコンソールアクセスを制限する](#)
- [プール内のすべての ID に対して特定のデータセットへのアクセスを許可する](#)

ポリシーに関するベストプラクティス

ID ベースのポリシーは、アカウント内で誰が Amazon Cognito リソースを作成、アクセス、または削除できるを決定します。これらのアクションでは、AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらはで使えます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイドの [AWS マネージドポリシー](#) または [ジョブ機能の AWS マネージドポリシー](#) を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイドの [IAM でのポリシーとアクセス許可](#) を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションがなどの特定のを通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイドの [IAM JSON ポリシー要素:条件](#) を参照してください。
- IAM アクセスアナライザーを使用して IAM ポリシーを検証し、安全で機能的な権限を確保する – IAM アクセスアナライザーは、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイドの [IAM Access Analyzer でポリシーを検証する](#) を参照してください。
- 多要素認証 (MFA) を要求する – IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーション

が呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイドの [MFA を使用した安全な API アクセス](#) を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイドの [IAM でのセキュリティのベストプラクティス](#) を参照してください。

Note

Amazon Cognito コンソールの元のバージョンと新しいバージョンでは、Amazon Cognito リソースを表示および変更するときの基本的な動作が異なります。条件 `aws:ViaAWSService` が `true` のときだけに `cognito-idp` サービスプレフィックスのアクションに許可を付与した場合、該当する IAM プリンシパルは、元のコンソールでは Amazon Cognito リソースに対して有効かもしれませんが、新しいコンソールでは使用できません。Amazon Cognito コンソールで作業を行うには、IAM ポリシーの Amazon Cognito アクセス許可に対して `aws:ViaAWSService` 条件を設定しないでください。

Amazon Cognito コンソールの使用

Amazon Cognito コンソールにアクセスするには、最小限の許可のセットが必要です。これらのアクセス許可により、の Amazon Cognito リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続き Amazon Cognito コンソールを使用できるようにするには、エンティティに Amazon Cognito ConsoleAccess または ReadOnly AWS 管理ポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「[ユーザーへのアクセス許可の追加](#)」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、

または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
  "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": "*"
    }
  ]
}
```

特定の ID プールにコンソールアクセスを制限する

JSON

```
{
  "Version": "2012-10-17",
```



```

    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "cognito-identity:ListIdentityPools"
        ],
        "Resource": "*"
      },
      {
        "Effect": "Allow",
        "Action": [
          "cognito-identity:*"
        ],
        "Resource": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-east-1:1a1a1a1a-ffff-1111-9999-12345678"
      },
      {
        "Effect": "Allow",
        "Action": [
          "cognito-sync:*"
        ],
        "Resource": "arn:aws:cognito-sync:us-east-1:111122223333:identitypool/us-east-1:1a1a1a1a-ffff-1111-9999-12345678"
      }
    ]
  }
}

```

プール内のすべての ID に対して特定のデータセットへのアクセスを許可する

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cognito-sync:ListRecords",
        "cognito-sync:UpdateRecords"
      ],

```



```
"Resource": "arn:aws:cognito-sync:us-east-1:111122223333:identitypool/us-east-1:1a1a1a1a-ffff-1111-9999-12345678/identity/*/dataset/UserProfile"
    }
  ]
}
```

Amazon Cognito のアイデンティティとアクセスのトラブルシューティング

以下の情報は、Amazon Cognito と IAM の使用時に発生する可能性がある一般的な問題の診断と修正に役立ちます。

トピック

- [Amazon Cognito でアクションを実行する権限がない](#)
- [iam:PassRole を実行する権限がありません](#)
- [管理者として Amazon Cognito へのアクセスを他のユーザーに許可したい](#)
- [AWS アカウント以外のユーザーに Amazon Cognito リソースへのアクセスを許可したい](#)

Amazon Cognito でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な `cognito-identity:GetWidget` アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
cognito-identity:GetWidget on resource: my-example-widget
```

この場合、`cognito-identity:GetWidget` アクションを使用して *my-example-widget* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、Amazon Cognito にロールを渡すことを許可するようにポリシーを更新する必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

次の例では、marymajor という名前の IAM ユーザーがコンソールを使用して Amazon Cognito でアクションを実行しようとした際に、エラーが発生しています。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

管理者として Amazon Cognito へのアクセスを他のユーザーに許可したい

他のユーザーが Amazon Cognito にアクセスできるようにするには、アクセスが必要なユーザーまたはアプリケーションに許可を付与する必要があります。AWS IAM アイデンティティセンターを使用してユーザーとアプリケーションを管理する場合は、アクセスレベルを定義する許可セットをユーザーまたはグループに割り当てます。アクセス許可セットは、ユーザーまたはアプリケーションに関連付けられている IAM ロールに自動的に IAM ポリシーを作成して割り当てます。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[アクセス許可セット](#)」を参照してください。

IAM アイデンティティセンターを使用していない場合は、アクセスが必要なユーザーまたはアプリケーション用に IAM エンティティ (ユーザーまたはロール) を作成する必要があります。次に、Amazon Cognito で適切なアクセス許可を付与するポリシーをエンティティにアタッチする必要があります。許可が付与されたら、ユーザーまたはアプリケーション開発者に認証情報を提供します。これらの認証情報を使用して AWS にアクセスします。IAM ユーザー、グループ、ポリシー、ア

クセス許可の作成の詳細については、「IAM ユーザーガイド」の「[IAM アイデンティティ](#)」と「[IAM のポリシーとアクセス許可](#)」を参照してください。

AWS アカウント以外のユーザーに Amazon Cognito リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon Cognito がこれらの機能をサポートしているかどうかについては、「[Amazon Cognito で IAM が機能する仕組み](#)」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「[所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する](#)」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「[サードパーティー AWS アカウント が所有する へのアクセスを提供する](#)」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの [外部で認証されたユーザー \(ID フェデレーション\) へのアクセスの許可](#) を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用法の違いについては、IAM ユーザーガイドの [IAM でのクロスアカウントのリソースへのアクセス](#) を参照してください。

Amazon Cognito のサービスリンクロールの使用

Amazon Cognito は AWS Identity and Access Management (IAM) [サービスにリンクされたロール](#)を使用します。サービスにリンクされたロールは、 がロール AWS のサービス を引き受けることを許可する信頼ポリシーを持つ一意のタイプの IAM ロールです。サービスにリンクされたロールは Amazon Cognito によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスリンクロールでは、必要な許可を手動で追加する必要がないため、Amazon Cognito の設定が簡単になります。Amazon Cognito は、そのサービスリンクロールの許可を定義します。別途定義

されている場合を除き、Amazon Cognito しかそのロールを引き受けることができません。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これは、Amazon Cognito リソースを保護します。リソースにアクセスするための許可を誤って削除できなくなるからです。

サービスにリンクされたロールをサポートする他のサービスについては、「[IAM と連携するAWS サービス](#)」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] (はい) リンクを選択します。

Amazon Cognito のサービスリンクロール許可

Amazon Cognito は、以下のサービスリンクロールを使用します。

- `AWSServiceRoleForAmazonCognitoIdpEmailService` – Amazon Cognito のユーザープールサービスが E メール送信に Amazon SES アイデンティティを使用できるようにします。
- `AWSServiceRoleForAmazonCognitoIdp` – Amazon Cognito のユーザープールが、Amazon Pinpoint プロジェクトのイベントを発行し、エンドポイントを設定できるようにします。

`AWSServiceRoleForAmazonCognitoIdpEmailService`

`AWSServiceRoleForAmazonCognitoIdpEmailService` サービスリンクロールは、ロールの引き受けについて以下のサービスを信頼します。

- `email.cognito-idp.amazonaws.com`

このロールの許可ポリシーは、Amazon Cognito が指定されたリソースで以下のアクションを実行できるようにします。

`AWSServiceRoleForAmazonCognitoIdpEmailService` に許可されるアクション:

- アクション: `ses:SendEmail` および `ses:SendRawEmail`
- リソース: *

このポリシーは、Amazon Cognito が指定されたリソースで以下のアクションを完了することを拒否します。

拒否されるアクション

- アクション: `ses:List*`
- リソース: `*`

これらの許可を使用すると、Amazon Cognito は、ユーザーへの E メール送信に Amazon SES で検証済みの E メールアドレスのみを使用することができます。Amazon Cognito は、ユーザーがユーザープールのクライアントアプリで特定のアクション (サインアップやパスワードのリセットなど) を実行するときに、ユーザーに E メールを送信します。

IAM エンティティ (ユーザー、グループ、ロールなど) がサービスリンクロールの作成、編集、削除を行うことを許可する許可を設定する必要があります。詳細については IAM ユーザーガイド の「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

AWSServiceRoleForAmazonCognitoIdp

サービスにリンクされたロール `AWSServiceRoleForAmazonCognitoIdp` は、以下のサービスを信頼してロールを引き受けます。

- `email.cognito-idp.amazonaws.com`

このロールの許可ポリシーは、Amazon Cognito が指定されたリソースで以下のアクションを実行できるようにします。

AWSServiceRoleForAmazonCognitoIdp に許可されるアクション

- アクション: `cognito-idp:Describe`
- リソース: `*`

この許可を使用すると、Amazon Cognito はユーザーに代わって Describe Amazon Cognito API オペレーションを呼び出すことができます。

Note

`createUserPoolClient` と `updateUserPoolClient` を使用して Amazon Cognito を Amazon Pinpoint に統合すると、リソースのアクセス許可がインラインポリシーとして SLR

に追加されます。インラインポリシーは、mobiletargeting:UpdateEndpoint および mobiletargeting:PutEvents 許可を提供します。これらの許可は、Amazon Cognito が、Cognito に統合された Pinpoint プロジェクトのためにイベントを発行し、エンドポイントを設定できるようにします。

Amazon Cognito のサービスリンクロールの作成

サービスリンクロールを手動で作成する必要はありません。Amazon SES 設定を使用して、AWS マネジメントコンソール、AWS CLI または Amazon Cognito API で E メール配信を処理するようにユーザープールを設定すると、Amazon Cognito によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要がある場合は同じ方法でアカウントにロールを再作成できます。Amazon SES 設定を使用してメール配信を処理するようにユーザープールを設定すると、Amazon Cognito がサービスリンクロールをもう一度作成します。

Amazon Cognito がこのロールを作成する前に、ユーザープールのセットアップに使用する IAM 許可に iam:CreateServiceLinkedRole アクションを含める必要があります。IAM での許可の更新に関する詳細については、IAM ユーザーガイドの「[IAM ユーザーの許可の変更](#)」を参照してください。

Amazon Cognito のサービスリンクロールの編集

AmazonCognitoIdp または AmazonCognitoIdpEmailService サービスにリンクされたロールは編集できません AWS Identity and Access Management。サービスリンクロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロール記述の編集はできます。詳細については、「IAM ユーザーガイド」の「[サービスリンクロールの編集](#)」を参照してください。

Amazon Cognito のサービスリンクロールの削除

サービスリンクロールを必要とする機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。ロールを削除すると、Amazon Cognito が積極的にモニタリングまたは維持しているエンティティのみを保持します。AmazonCognitoIdp または AmazonCognitoIdpEmailService サービスリンクロールを削除する前に、このロールを使用する各ユーザープールに対して以下のいずれかを実行する必要があります。

- ユーザープールを削除する。

- ユーザープールの E メール設定を更新して、デフォルトの E メール機能を使用する。デフォルト設定では、サービスリンクロールが使用されません。

ロールを使用するユーザープール AWS リージョン を使用して、各 で アクションを実行することを忘れないでください。

Note

リソースを削除しようとするときに Amazon Cognito サービスがロールを使用している場合は、削除が失敗する可能性があります。失敗した場合は、数分待ってから操作を再試行してください。

Amazon Cognito ユーザープールを削除する

1. にサインイン AWS マネジメントコンソール し、 で Amazon Cognito コンソールを開きます <https://console.aws.amazon.com/cognito>。
2. [Manage User Pools] (ユーザープールの管理) をクリックします。
3. [Your User Pools] (ユーザープール) ページで、削除するユーザープールを選択します。
4. [Delete pool] (プールの削除) をクリックします。
5. [Delete user pool] (ユーザープールの削除) ウィンドウで「**delete**」と入力し、[Delete pool] (プールの削除) をクリックします。

デフォルトの E メール機能を使用するために Amazon Cognito ユーザープールを更新する

1. にサインイン AWS マネジメントコンソール し、 で Amazon Cognito コンソールを開きます <https://console.aws.amazon.com/cognito>。
2. [Manage User Pools] (ユーザープールの管理) をクリックします。
3. [Your User Pools] (ユーザープール) ページで、更新するユーザープールを選択します。
4. 左側のナビゲーションメニューで [Message customizations] (メッセージのカスタマイズ) をクリックします。
5. [Do you want to send emails through your Amazon SES Configuration?] (Amazon SES の設定を通じて E メールを送信しますか?) で [No - Use Cognito (Default)] (いいえ - Cognito を使用します (デフォルト)) を選択します。

6. E メールアカウントオプションの設定を終了したら、[Save changes] (変更の保存) をクリックします。

サービスリンクロールを IAM で手動削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AmazonCognitoIdp または AmazonCognitoIdpEmailService サービスにリンクされたロールを削除します。詳細については、「IAM ユーザーガイド」の「[サービスにリンクされたロールの削除](#)」を参照してください。

Amazon Cognito サービスリンクロールがサポートされるリージョン

Amazon Cognito は、サービスが利用可能なすべての AWS リージョン でサービスにリンクされたロールをサポートします。詳細については、「[AWS リージョン およびエンドポイント](#)」を参照してください。

Amazon Cognito でのロギングとモニタリング

モニタリングは、Amazon Cognito およびその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。現在 Amazon Cognito では、組織と、その組織内で発生するアクティビティを監視することができるよう、以下の AWS のサービスをサポートしています。

- AWS CloudTrail – CloudTrail を使用すると、Amazon Cognito コンソールと Amazon Cognito API オペレーションへのコード呼び出しから API コールをキャプチャできます。例えば、ユーザーが認証すると、CloudTrail はリクエストの IP アドレス、リクエストの実行者、および実行日時などの詳細を記録できます。
- Amazon CloudWatch Logs — CloudWatch Logs を使用すると、ユーザーアクティビティの詳細なログをロググループに送信できます。例えば、ユーザーアクティビティの詳細ログを確認して、ユーザーへの E メールや SMS メッセージの配信のトラブルシューティングを行うことができます。
- Amazon CloudWatch メトリクス – CloudWatch メトリクスでは、監視、報告、およびイベントが発生した場合のほぼリアルタイムでの自動アクションの実行が可能です。例えば、提供されたメトリクスで CloudWatch ダッシュボードを作成して Amazon Cognito ユーザープールを監視する、または提供されたメトリクスで CloudWatch アラームを作成して、設定されたしきい値の超過を通知することができます。

- Amazon CloudWatch Logs Insights – CloudWatch Logs Insights では、Amazon Cognito CloudTrail ログファイルをモニタリングするため、CloudWatch にイベントを送信するよう CloudTrail を設定できます。

トピック

- [コストのモニタリングと管理](#)
- [Amazon Cognito ユーザープールからのログのエクスポート](#)
- [CloudWatch と Service Quotas でのクォータと使用状況の追跡](#)
- [Amazon Cognito ログイン AWS CloudTrail](#)

コストのモニタリングと管理

他のと同様に AWS のサービス、Amazon Cognito の設定と使用状況が AWS 請求に与える影響を理解することが重要です。ユーザープールを本番環境にデプロイするための準備の一環として、アクティビティとリソースの消費のモニタリングと保護を設定します。どこを見ればよいか、またどのアクションが追加コストを発生させるかがわかれば、請求書を見て驚くことがないように予防措置を講じることができます。

Amazon Cognito は、使用量の次のディメンションに対して課金します。

- ユーザープールの月間アクティブユーザー (MAU) — 料金は[機能プラン](#)に応じて異なります
- OIDC または SAML フェデレーションでサインインしたユーザープール MAU
- アクティブユーザープールアプリケーションクライアントと、クライアント認証情報の付与によるマシンツーマシン (M2M) 認証のリクエストボリューム
- 一部のカテゴリのユーザープール API のデフォルトクォータ値を超える使用量の購入

さらに、E メールメッセージ、SMS メッセージ、Lambda トリガーなどのユーザープールの機能によって、依存サービスにコストが発生する可能性があります。完全な概要については、「[Amazon Cognito 料金](#)」を参照してください。

コストの表示と予測

製品の起動や、新しいユーザーベースへのオープンなどの大量のイベントは、MAU 数を増やし、コストに影響を与える可能性があります。新しいユーザー数を事前に見積もり、アクティビティが発生したときに監視します。クォータ値を追加購入してボリュームに対応させたり、追加のセキュリティ対策でボリュームを制御したりできます。

[AWS Billing and Cost Management コンソール](#)で AWS コストを表示およびレポートできます。Amazon Cognito の最新の請求額は、[請求と支払い] セクションで確認できます。[請求書] の [サービス別料金] で、Cognito をフィルタリングして使用状況を表示します。詳細については AWS Billing ユーザーガイドの [請求書の表示](#) を参照してください。

API リクエストレートをモニタリングするには、Service Quotas コンソールで [使用状況] メトリクスを確認します。例えば、クライアント認証情報リクエストは [ClientAuthentication リクエストのレート] として表示されます。請求書では、これらのリクエストは、それらを生成したアプリケーションクライアントに関連付けられています。この情報により、[マルチテナントアーキテクチャ](#) のテナントに公平にコストを割り当てることができます。

一定期間の M2M リクエストの数を取得するには、分析のために [AWS CloudTrail イベントを CloudWatch Logs](#) に送信することもできます。クライアント認証情報のイベント付与を使用して Token_POST イベントのために CloudTrail イベントにクエリを設定します。次の CloudWatch Insights クエリはログを返します。

```
filter eventName = "Token_POST" and @message like '"grant_type":["client_credentials"]'
| stats count(*)
```

のコスト管理

Amazon Cognito は、ユーザー数、機能の使用状況、リクエストボリュームに基づいて請求します。Amazon Cognito のコストを管理するためのヒントを以下に紹介します。

非アクティブなユーザーをアクティブ化しない

ユーザーをアクティブにする一般的なオペレーションは、サインイン、サインアップ、パスワードのリセットです。詳細なリストについては、「[月次のアクティブユーザー](#)」を参照してください。Amazon Cognito は、非アクティブなユーザーを請求にカウントしません。ユーザーをアクティブに設定するオペレーションは避けてください。ユーザーに対してクエリを行うには、[AdminGetUser](#) API オペレーションの代わりに、[ListUsers](#) オペレーションを使用します。非アクティブなユーザーに対して、ユーザープールオペレーションの大量の管理テストを実行しないでください。

フェデレーションユーザーのリンク

SAML 2.0 または OpenID Connect (OIDC) ID プロバイダーでサインインするユーザーは、[ローカルユーザー](#) よりもコストが高くなります。[これらのユーザーをローカルユーザープロフィールにリンク](#) できます。リンクされたユーザーは、フェデレーションユーザーに含まれる属性とアクセスを使用

して、ローカルユーザーとしてサインインできます。SAML または OIDC IdPs のユーザーが、1 か月間、リンクされたローカルアカウントでのみサインインすると、ローカルユーザーとして課金されます。

リクエストレートを管理する

ユーザープールがクォータ値の上限に近づいている場合は、ボリュームを処理するための追加の容量の購入を検討してください。アプリケーション内のリクエストの量を減らせる場合があります。詳細については、「[クォータ制限のリクエストレートを最適化する](#)」を参照してください。

必要な場合にのみ新しいトークンをリクエストする

クライアント認証情報の付与によるマシンツーマシン (M2M) 認証は、大量のトークンリクエストに達する可能性があります。新しいトークンリクエストはどれも、リクエストレートのクォータ量と請求の大きさに影響します。コストを最適化するには、アプリケーションの設計で、トークンの有効期限設定とトークン処理を含めます。

- [アクセストークンをキャッシュ](#)して、アプリケーションが新しいトークンをリクエストしたときに、以前に発行されたトークンのキャッシュされたバージョンを受け取るようにします。この方法を実装すると、キャッシュプロキシは、以前に取得したトークンの有効期限を認識せずにアクセストークンをリクエストするアプリケーションに対するガードとして機能します。キャッシュトークンは、Lambda 関数や Docker コンテナなどの有効期間の短いマイクロサービスに最適です。
- トークンの有効期限を考慮したトークン処理メカニズムをアプリケーションに実装します。以前のトークンの有効期限が切れるまで、新しいトークンをリクエストしないでください。ベストプラクティスとしては、トークンの有効期間の約 75% でトークンを更新します。このプラクティスでは、トークンの有効期間を最大化しながら、アプリケーションのユーザー継続性を確保できます。

各アプリケーションの機密性と可用性のニーズを評価し、適切な有効期間でアクセストークンを発行するようにユーザープールアプリケーションクライアントを設定します。カスタムのトークン有効期間は、認証情報に対するリクエストの頻度を永続的に管理できる、存続期間の長い API やサーバーに最適です。

AdminGetUser よりも ListUsers を使用する

ユーザープール内のユーザーの属性をクエリするには、[ListUsers](#) API オペレーションおよび関連する [SDK](#) メソッドをできるだけ使用します。[AdminGetUser](#) を使用すると、ユーザーはその月のアクティブユーザーとしてマークされ、月間アクティブユーザー (MAU) 数に加算されて、ユーザープールの請求額の計算に使用されます。

未使用のクライアント認証情報アプリケーションクライアントを削除する

M2M 認証は、トークンリクエストのレートと、クライアント認証情報が付与されるアプリケーションクライアントの数という 2 つの要素に基づいて請求されます。M2M 認証用のアプリケーションクライアントが使用されていない場合は、それらを削除するか、クライアント認証情報を発行するための認証を削除します。アプリケーションクライアント設定の管理の詳細については、「[アプリケーションクライアントによるアプリケーション固有の設定](#)」を参照してください。

機能プランを管理する

ユーザープールで[機能プラン](#)を選択すると、請求レートはユーザープール内のすべての MAU に適用されます。高レベルの機能プランに付属する機能を必要としないユーザーがいる場合は、そのユーザーを別のユーザープールに分離します。

Amazon Cognito ユーザープールからのログのエクスポート

CloudWatch ロググループなど AWS のサービス、追加のアクティビティの詳細ログを別のアクティビティに送信するようにユーザープールを設定できます。これらのログは のログよりも細かく AWS CloudTrail、ユーザープールのトラブルシューティングや[脅威に対する保護](#)によるユーザーサインインアクティビティの分析に役立ちます。SMS および E メール通知エラーのログをストリーミングする場合、ユーザープールは ERROR レベルのログを CloudWatch ロググループに送信します。ユーザーサインインアクティビティのログをストリーミングする場合、ユーザープールは INFO レベルのログをロググループ、Amazon Data Firehose ストリーム、または Amazon S3 バケットに送信します。ユーザープールで両方のオプションを組み合わせることができます。

トピック

- [ログエクスポートについて知っておくべきこと](#)
- [E メールおよび SMS メッセージ配信エラーのエクスポート](#)
- [脅威保護ユーザーアクティビティログのエクスポート](#)

ログエクスポートについて知っておくべきこと

コストへの影響

Amazon Data Firehose、Amazon S3、CloudWatch Logs では、データインGESTと取得にコストが発生します。ログ記録設定が AWS 請求書に影響する場合があります。詳細については次を参照してください:

- Amazon CloudWatch の料金の [Vended Logs](#)。
- [Amazon Data Firehose の料金](#)
- [Amazon S3 の価格設定](#)

ユーザーアクティビティログのエクスポートには、セキュリティ評価が含まれています。これはユーザープールの[脅威保護](#)の1つの機能です。Amazon Cognito は、脅威保護が監査専用モードまたはフル機能モードであり、ユーザープールでプラス[機能プラン](#)が有効になっている場合にのみ、これらのログを生成します。

ユーザーアクティビティログは INFO レベル

エクスポートされるユーザーアクティビティログは INFO エラーレベルに限られ、認証アクティビティの統計およびセキュリティ分析用の情報を提供します。WARNING および ERROR エラーレベル (スロットリングエラーなど) のメッセージは、エクスポートされるログに含まれません。

ベストエフォート配信

Amazon Cognito からのログの配信はベストエフォートに基づきます。ユーザープールが配信するログの量と、CloudWatch Logs、Amazon S3、Firehose のサービスクォータは、ログの配信に影響を与える可能性があります。

既存の外部ログは影響を受けません

これらのログオプションは、ユーザープールの以下のログ機能を置き換えたり変更したりするものではありません。

1. サインアップやサインインなどの日常的なユーザーアクティビティの CloudTrail ログ。
2. CloudWatch メトリクスを使用した大規模なユーザーアクティビティの分析。

別に、CloudWatch Logs で [ユーザープールのインポート結果を CloudWatch コンソールに表示](#) や [Lambda トリガーを使用したユーザープールワークフローのカスタマイズ](#) のログを参照することもできます。Amazon Cognito と Lambda は、ユーザーアクティビティログ用に指定したロググループとは異なるロググループに、これらのログを保存します。

ユーザープールにのみ適用されます

アイデンティティプールにログエクスポート機能はありません。

ユーザーアクセス許可とサービスにリンクされたロールが必要です

ログのエクスポートを設定する AWS プリンシパルには、以下のトピックで説明するように、ターゲットリソースを変更するアクセス許可が必要です。Amazon Cognito はユーザーに代わっ

で[サービスにリンクされたロール](#)を作成し、ターゲットリソースにログを配信するロールを引き受けます。

Amazon Cognito からログを送信するための認可モデルの詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[AWS のサービスからのロギングを有効にする](#)」を参照してください。

ログレベルはログタイプに排他的です

メッセージ配信ログは、userNotification タイプと ERROR エラーレベルのログです。高度なセキュリティユーザーアクティビティログは、userAuthEvents タイプと INFO エラーレベルのログです。LogConfigurations の 2 つのメンバーを組み合わせることができます。1 つは CloudWatch Logs の userNotification 用メンバーで、もう 1 つは Firehose、Amazon S3、または CloudWatch Logs の userAuthEvents 用メンバーです。

ユーザーアクティビティログを複数の送信先に送信できません。CloudWatch Logs 以外の送信先にユーザー通知ログを送信できません。

さまざまな設定オプション

ユーザー通知ログは、Amazon Cognito ユーザープール API または AWS SDK でのみ設定できます。API または Amazon Cognito コンソールで、高度なセキュリティユーザーアクティビティログを設定できます。両方を設定するには、[SetLogDeliveryConfiguration](#) のリクエスト例で示されているように API を使用します。

大規模なリソースベースのポリシーで必要な追加設定

サイズが 5,120 文字を超えるリソースポリシーを持つロググループにログを送信するには、/aws/vendedlogs で始まるパスでロググループを設定します。詳細については、「[特定の AWS サービスからのログ記録の有効化](#)」を参照してください。

Amazon S3 でのフォルダの自動作成

Amazon S3 バケットに脅威保護ログをエクスポートするように設定すると、Amazon Cognito はバケット内に AWSLogs フォルダを作成する場合があります。このフォルダはすべての場合に作成されるわけではなく、作成されなくても設定は成功することがあります。

E メールおよび SMS メッセージ配信エラーのエクスポート

E メールおよび SMS メッセージ配信エラーの場合、ユーザープールからエラーレベルのユーザー通知ログを配信できます。この機能を有効にすると、Amazon Cognito からログを送信する先のロググループを選択できます。ユーザー通知のログ記録は、ユーザープールが Amazon SNS と Amazon

SES で配信した E メールや SMS メッセージのステータスを確認する場合に便利です。このログのエクスポートオプションは、[ユーザーアクティビティのエクスポート](#)とは異なり、プラス機能プランを必要としません。

[SetLogDeliveryConfiguration](#) API リクエストで Amazon Cognito ユーザープール API を使用して詳細な通知ログを設定できます。ユーザープールのログ記録設定は、[GetLogDeliveryConfiguration](#) API リクエストで確認できます。リクエスト本文の例を次に示します。

```
{
  "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"
}
```

これらのリクエストは、以下のアクセス許可を持つ AWS 認証情報を使用して認可する必要があります。

JSON

```
{
  "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"
    }
  ]
}

```

ユーザープールのイベントの例を次に示します。このログスキーマは変更される可能性があります。一部のフィールドは NULL 値がログとして記録される場合があります。

```

{
  "eventTimestamp": "1687297330677",
  "eventSource": "USER_NOTIFICATION",
  "logLevel": "ERROR",
  "message": {
    "details": "String"
  },
  "logSourceId": {
    "userPoolId": "String"
  }
}

```



```
}
```

脅威保護ユーザーアクティビティログのエクスポート

プラス機能プランと脅威保護を備えたユーザープールは、ユーザーアクティビティイベントとして、ユーザープールでのユーザーのサインイン、サインアウト、その他の認証オペレーションに関する詳細とセキュリティ評価をログを記録します。独自のログ管理システムでユーザーアクティビティログを確認したり、アーカイブを作成したりできます。Amazon CloudWatch Logs ロググループ、Amazon Data Firehose ストリーム、Amazon Simple Storage Service (Amazon S3) バケットを使用できます。そこから、このデータを、運用プロセスに適合する方法でデータを分析、正規化、またはその他の方法で処理する他のシステムに取り込むことができます。このタイプのデータをエクスポートするには、ユーザープールでプラス機能プランを使用しており、ユーザープールで[脅威保護](#)がアクティブになっている必要があります。

これらのユーザーアクティビティログの情報をを使用して、ユーザーサインインとアカウント管理アクティビティのプロファイルを表示できます。デフォルトでは、Amazon Cognito はこれらのイベントをユーザープールに基づくストレージにキャプチャします。次の例は、サインインし、リスク要因がないと評価されたユーザーのイベント例です。この情報は、AdminListUserAuthEvents API オペレーションで取得できます。以下は、その出力例です。

```
{
  "AuthEvents": [
    {
      "EventId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
      "EventType": "SignIn",
      "CreationDate": "2024-06-27T10:49:59.139000-07:00",
      "EventResponse": "Pass",
      "EventRisk": {
        "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"
}
```

Amazon Cognito コンソール内で、または [SetLogDeliveryConfiguration](#) API オペレーションを使用して、ユーザーアクティビティのログエクスポートをアクティブ化できます。

AWS マネジメントコンソール

1. 使用するバケットがまだない場合は、[S3 バケット](#)、[Firehose ストリーム](#)、または [CloudWatch ロググループ](#)を作成します。
2. [Amazon Cognito コンソール](#)にサインインします。
3. [User Pools] (ユーザープール) を選択します。
4. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
5. [高度なセキュリティ] タブを選択します。[ユーザーアクティビティログをエクスポートする] を見つけて [編集] を選択します
6. [ログ記録ステータス] で、[ユーザーアクティビティログのエクスポートをアクティブ化] の横にあるチェックボックスをオンにします。
7. ログ記録先で、ログ AWS のサービス を処理する を選択します: CloudWatch ロググループ、Amazon Data Firehose ストリーム、または S3 バケット。
8. 選択すると、対応するリソースタイプがリソースセレクターに入力されます。リストからロググループ、ストリーム、またはバケットを選択します。作成ボタンを選択して、AWS マネジメントコンソール 選択したサービスの に移動し、新しいリソースを作成することもできます。
9. [変更を保存] を選択します。

API

ユーザーアクティビティログの送信先のタイプを 1 つ選択します。

以下は、Firehose ストリームをログの送信先として設定する `SetLogDeliveryConfiguration` リクエスト本文の例です。

```
{
```

```
"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"  
}
```

以下は、Amazon S3 バケットをログの送信先として設定する SetLogDeliveryConfiguration リクエスト本文の例です。

```
{  
  "LogConfigurations": [  
    {  
      "EventSource": "userAuthEvents",  
      "S3Configuration": {  
        "BucketArn": "arn:aws:s3:::amzn-s3-demo-logging-bucket"  
      },  
      "LogLevel": "INFO"  
    },  
  ],  
  "UserPoolId": "us-west-2_EXAMPLE"  
}
```

以下は、CloudWatch ロググループをログの送信先として設定する SetLogDeliveryConfiguration リクエスト本文の例です。

```
{  
  "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"  
  }  
}
```

ログ配信を設定するユーザーは、ユーザープール管理者で、かつ、次の追加のアクセス許可を持っている必要があります。

Amazon S3

JSON

```
{  
  "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

JSON

```
{
  "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

JSON

```
{
  "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": "*"
    }
  ]
}
```

ユーザープールのイベントの例を次に示します。このログスキーマは変更される可能性があります。一部のフィールドは NULL 値がログとして記録される場合があります。

```
{
  "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": "",
    "hasContextData": "true"
  },
  "logSourceId": {
    "userPoolId": "us-west-2_EXAMPLE"
  }
}
```

CloudWatch と Service Quotas でのクォータと使用状況の追跡

Amazon CloudWatch または Service Quotas を使用して、Amazon Cognito ユーザープールをモニタリングできます。Service Quotas でアイデンティティプールの使用状況をモニタリングすることもできます。CloudWatch は、raw データを収集して、それを読み取り可能なほぼリアルタイムのメトリクスに変換します。CloudWatch では、特定のしきい値を監視するアラームを設定し、これらのしきい値に到達したときに通知を送信、またはアクションを実行することができます。サービスクォータ向けの CloudWatch アラームを作成するには、「[CloudWatch アラームを作成する](#)」を参照してください。Amazon Cognito のメトリクスは 5 分間隔で利用できます。CloudWatch での保持期間の詳細については、「[Amazon CloudWatch のよくある質問](#)」ページを参照してください。

Service Quotas を使用して、Amazon Cognito ユーザープールとアイデンティティプールのクォータの使用状況を表示および管理できます。Service Quotas コンソールには、サービスクォータの表示、サービスクォータ引き上げのリクエスト、および現在の使用状況の表示という 3 つの機能があ

ります。最初の機能を使用してクォータを表示し、クォータが調整可能かどうかを確認できます。2番目の機能を使用して、Service Quotas の引き上げをリクエストできます。最後の機能は、クォータの使用率の表示に使用できます。この機能を利用できるのは、アカウントがしばらくの間アクティブな状態であった場合のみです。Service Quotas コンソールでのクォータの表示に関する詳細については、「[Viewing service quotas](#)」を参照してください。

Note

Amazon Cognito のメトリクスは 5 分間隔で利用可能になります。CloudWatch での保持期間の詳細については、「[Amazon CloudWatch のよくある質問](#)」ページを参照してください。

CloudWatch クロスアカウントオブザーバビリティでモニタリングアカウントとして AWS アカウント設定されたにサインインしている場合、そのモニタリングアカウントを使用してサービスクォータを視覚化し、そのモニタリングアカウントにリンクされているソースアカウントのメトリクスのアラームを設定できます。詳細については、「[CloudWatch のクロスアカウントオブザーバビリティ](#)」を参照してください。

トピック

- [CloudWatch のユーザープールメトリクス](#)
- [Service Quotas のメトリクス](#)

CloudWatch のユーザープールメトリクス

ユーザープールは、ユーザーアクティビティ統計をメトリクスとして CloudWatch に報告します。CloudWatch から、ユーザープール内の認証アクティビティの量とクォータ使用率を分析できます。これらのメトリクスの情報を使用して、注目すべきイベントのアラームを設定し、必要に応じてユーザープールの設定を調整できます。ユーザーアクティビティのログ記録にユーザープール内のユーザーアクティビティの詳細なレコードがある場合、CloudWatch メトリクスには統計とパフォーマンス指標が集約されています。

以下の表には、Amazon Cognito ユーザープールに使用できるメトリクスがリストされています。Amazon Cognito は、メトリクスを名前空間 AWS/Cognito および AWS/Usage に発行します。詳細については、「Amazon CloudWatch ユーザーガイド」の「[名前空間](#)」を参照してください。

クォータと使用状況の追跡の詳細については、「[クォータの使用状況を追跡する](#)」および「[月間アクティブユーザー数 \(MAU\) を追跡する](#)」を参照してください。

 Note

過去 2 週間に新しいデータポイントがなかったメトリクスは、コンソールに表示されません。また、コンソールの [All metrics] (すべてのメトリクス) タブにある検索ボックスにこれらのメトリクス名またはディメンション名を入力しても表示されません。さらに、list-metrics コマンドの結果にも返されません。これらのメトリクスを取得する最善の方法は、AWS CLI の get-metric-data または get-metric-statistics コマンドを使用することです。

メトリクス	説明	名前空間
SignUpSuccesses	Amazon Cognito ユーザープールに対して正常に行われたユーザー登録リクエストの合計数を提供します。正常に行われたユーザー登録リクエストは 1 の値を生成し、失敗したリクエストは 0 の値を生成します。スロットリングされたリクエストも失敗したリクエストと見なされるため、スロットリングされたリクエストも 0 の数を生成します。 正常に行われたユーザー登録リクエストの割合を確認するには、このメトリクスの Average 統計を使用します。ユーザー登録リクエストの合計数を計上するには、このメトリクスの Sample Count 統計を使用します。正常に行われたユーザー登録リクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。失敗したユー	AWS/Cognito

メトリクス	説明	名前空間
	ザー登録リクエストの合計数を計上するには、CloudWatch Math 式を使用して、Sample Count 統計から Sum 統計を減算します。 このメトリクスは、各ユーザープールクライアントのユーザープールごとにパブリッシュされます。ユーザー登録が管理者によって実行された場合、メトリクスはユーザープールクライアントを Admin とした形でパブリッシュされます。 このメトリクスは、ユーザーのインポートおよびユーザーの移行には生成されません。 メトリクスディメンション: UserPool、UserPoolClient 単位: 数	

メトリクス	説明	名前空間
SignUpThrottles	Amazon Cognito ユーザープールに対して行われたユーザー登録リクエストで、スロットリングされたものの合計数を提供します。ユーザー登録リクエストがスロットリングされるたびに、1 の数が発行されます。 スロットリングされたユーザー登録リクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。 このメトリクスは、各クライアントのユーザープールごとにパブリッシュされます。スロットリングされたリクエストが管理者によって行われたものである場合、メトリクスはユーザープールクライアントを Admin とした形でパブリッシュされます。 メトリクスディメンション: UserPool、UserPoolClient 単位: 数	AWS/Cognito

メトリクス	説明	名前空間
SignInSuccesses	Amazon Cognito ユーザープールに対して正常に行われたユーザー認証リクエストの合計数を提供します。認証トークンがユーザーに発行されると、ユーザー認証は正常に行われたとみなされます。認証が正常に行われると 1 の値が生成され、リクエストが失敗すると 0 の値が生成されます。スロットリングされたリクエストも失敗したリクエストと見なされるため、スロットリングされたリクエストも 0 の数を生成します。 正常に行われたユーザー認証リクエストの割合を調べるには、このメトリクスの Average 統計を使用します。ユーザー認証リクエストの合計数を計上するには、このメトリクスの Sample Count 統計を使用します。正常に行われたユーザー認証リクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。失敗したユーザー認証リクエストの合計数を計上するには、CloudWatch Math 式を使用して、Sample Count 統計から Sum 統計を減算します。	AWS/Cognito

メトリクス	説明	名前空間
	このメトリクスは、各クライアントのユーザープールごとにパブリッシュされます。リクエストで無効なユーザープールクライアントが提供された場合、メトリクスの対応するユーザープールクライアント値には、リクエストで送信された実際の無効な値ではなく、固定値の <code>Invalid</code> が含まれます。 このメトリクスには Amazon Cognito トークンを更新するリクエストが含まれないことに注意してください。Refresh トークン統計の提供には、別のメトリクスがあります。 メトリクスディメンション: <code>UserPool</code>、<code>UserPoolClient</code> 単位: 数	

メトリクス	説明	名前空間
SignInThrottles	Amazon Cognito ユーザープールに対して行われたユーザー認証リクエストで、スロットリングされたものの合計数を提供します。認証リクエストがスロットリングされるたびに、1 の数が発行されます。 スロットリングされたユーザー認証リクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。 このメトリクスは、各クライアントのユーザープールごとにパブリッシュされます。リクエストで無効なユーザープールクライアントが提供された場合、メトリクスの対応するユーザープールクライアント値には、リクエストで送信された実際の無効な値ではなく、固定値の Invalid が含まれます。 このメトリクスに Amazon Cognito トークンを更新するリクエストは含まれません。Refresh トークン統計の提供には、別のメトリクスがあります。 メトリクスディメンション: UserPool、UserPoolClient	AWS/Cognito

メトリクス	説明	名前空間
	単位: 数	

メトリクス	説明	名前空間
TokenRefreshSuccesses	Amazon Cognito ユーザープールに対して行われた Amazon Cognito を更新するリクエストで、正常に行われたものの合計数を提供します。Amazon Cognito トークンの更新リクエストが正常に行われると 1 の値が生成され、失敗したリクエストには 0 の値が生成されます。スロットリングされたリクエストも失敗したリクエストと見なされるため、スロットリングされたリクエストも 0 の数を生成します。 正常に行われた Amazon Cognito トークンの更新リクエストの割合を調べるには、このメトリクスの Average 統計を使用します。Amazon Cognito トークンを更新するリクエストの合計数を計上するには、このメトリクスの Sample Count 統計を使用します。正常に行われた Amazon Cognito トークンの更新リクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。失敗した Amazon Cognito トークンの更新リクエストの合計数を計上するには、CloudWatch Math 式を使用して、Sample Count 統計から Sum 統計を減算します。	AWS/Cognito

メトリクス	説明	名前空間
	このメトリクスは、ユーザープールクライアントごとにパブリッシュされます。リクエストに無効なユーザープールクライアントがある場合、ユーザープールクライアントの値には固定値の <code>Invalid</code> が含まれます。 メトリクスディメンション: <code>UserPool</code>、<code>UserPoolClient</code> 単位: 数	

メトリクス	説明	名前空間
TokenRefreshThrottles	Amazon Cognito ユーザープールに対して行われた Amazon Cognito トークンを更新するリクエストで、スロットリングされたものの合計数を提供します。Amazon Cognito トークンの更新リクエストがスロットリングされるたびに、1 の数が発行されます。 Amazon Cognito トークンのスロットリングされた更新リクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。 このメトリクスは、各クライアントのユーザープールごとにパブリッシュされます。リクエストで無効なユーザープールクライアントが提供された場合、メトリクスの対応するユーザープールクライアント値には、リクエストで送信された実際の無効な値ではなく、固定値の Invalid が含まれます。 メトリクスディメンション: UserPool、UserPoolClient 単位: 数	AWS/Cognito

メトリクス	説明	名前空間
FederationSuccesses	Amazon Cognito ユーザープールに対して正常に行われた ID フェデレーションリクエストの合計数を提供します。ID フェデレーションは、Amazon Cognito がユーザーに認証トークンを発行した時点で成功とみなされます。ID フェデレーションリクエストが正常に行われると 1 の値が生成され、失敗すると 0 の値が生成されます。スロットリングされたリクエストや、認可コードは生成されるがトークンは生成されないリクエストでは、値が 0 になります。 正常に行われた ID フェデレーションリクエストの割合を調べるには、このメトリクスの Average 統計を使用します。ID フェデレーションリクエストの合計数を計上するには、このメトリクスの Sample Count 統計を使用します。正常に行われた ID フェデレーションリクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。失敗した ID フェデレーションリクエストの合計数を計上するには、CloudWatch Math 式を使用して、Sample Count 統計から Sum 統計を減算します。	AWS/Cognito

メトリクス	説明	名前空間
	メトリクスディメンション:UserPool、UserPoolClient、IdentityProvider 単位: 数	
FederationThrottles	Amazon Cognito ユーザープールに対して行われた ID フェデレーションリクエストで、スロットリングされたものの合計数を提供します。ID フェデレーションリクエストがスロットリングされるたびに、1 の数が発行されます。 スロットリングされた ID フェデレーションリクエストの合計数を計上するには、このメトリクスの Sum 統計を使用します。 メトリクスディメンション:UserPool、UserPoolClient、IdentityProvider 単位: 数	AWS/Cognito

メトリクス	説明	名前空間
CallCount	顧客が実行したコールで、カテゴリに関連するものの合計数を提供します。このメトリクスには、スロットリングコール、失敗したコール、正常に行われたコールなどのすべてのコールが含まれます。 カテゴリクォータは、AWSアカウントとリージョンのすべてのユーザープールにわたって、アカウントごとに適用されます。 カテゴリ内のコールの合計数は、このメトリクスの Sum 統計を使用して計上できます。 メトリクスディメンション: Type、Resource、Class 単位: 数	AWS/Usage

メトリクス	説明	名前空間
ThrottleCount	スロットルされたコールで、カテゴリに関連するものの合計数を提供します。 このメトリクスは、アカウントレベルでパブリッシュされます。 カテゴリ内のコールの合計数は、このメトリクスの Sum 統計を使用して計上できます。 メトリクスディメンション: Type、Resource、Class 単位: カウント	AWS/Usage

脅威保護メトリクスの表示

ユーザープールが発行するメトリクスには、脅威保護設定がユーザー認証アクティビティに与える影響に関する統計情報が含まれています。漏えいした認証情報でサインインしようとしているユーザーの数を知りたい場合があります。また、一定レベルのリスクがあると評価されたサインインアクティビティの割合も確認できます。Amazon Cognito は、脅威保護機能のメトリクスを Amazon CloudWatch のアカウントに発行します。Amazon Cognito は脅威保護のメトリクスを、リスクレベル別およびリスクエスレベル別にグループ化します。

リスク分析にコンテキストを追加するには、ユーザープールで、またはエクスポートされたデータソースで、[個々のユーザーサインイン試行に関する情報を表示](#)できます。

CloudWatch コンソールでメトリクスを表示する

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで [Metrics] (メトリクス) を選択します。
3. Amazon Cognito を選択します。
4. [By Risk Classification] (リスク分類別) などの集約されたメトリクスのグループを選択します。

5. [All metrics] (すべてのメトリクス) タブに、選択したグループのメトリクスが一覧表示されます。以下の操作を行うことができます。
- テーブルを並べ替えるには、列見出しを使用します。
 - メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。すべてのメトリクスを選択するには、テーブルの見出し行にあるチェックボックスを選択します。
 - リソースでフィルターするには、リソース ID を選択し、[Add to search] (検索に追加) を選択します。
 - メトリクスでフィルターするには、メトリクス名を選択し、[Add to search] (検索に追加) を選択します。

メトリクス	説明	メトリクスディメンション	名前空間
CompromisedCredentialRisk	Amazon Cognito が侵害された認証情報を検知したリクエスト。	操作: デイメンションは PasswordChange 、SignIn、または SignUp タイプの操作のみです。 UserPoolId: ユーザープールの識別子。 RiskLevel: 高 (デフォルト)、中、低。	AWS/Cognito
AccountTakeOverRisk	Amazon Cognito がアカウントの乗っ取りリスクを検知したリクエスト。	操作: デイメンションは PasswordChange 、SignIn、または SignUp タイプの操作のみです。 UserPoolId: ユーザープールの識別子。	AWS/Cognito

メトリクス	説明	メトリクスディメンション	名前空間
		RiskLevel: 高、中、低。	
OverrideBlock	Amazon Cognito がデベロッパーから提供された設定を理由にブロックしたリクエスト。	操作: デイメンションは PasswordChange 、SignIn、または SignUp タイプの操作のみです。 UserPoolId: ユーザープールの識別子。 RiskLevel: 高、中、低。	AWS/Cognito
Risk	Amazon Cognito がリスクありとしてマークしたリクエスト。	操作: PasswordChange 、SignIn、または SignUp などの操作タイプ。 UserPoolId: ユーザープールの識別子。	AWS/Cognito
NoRisk	Amazon Cognito がリスクを識別しなかったリクエスト。	操作: PasswordChange 、SignIn、または SignUp などの操作タイプ。 UserPoolId: ユーザープールの識別子。	AWS/Cognito

Amazon Cognito は、CloudWatch で即座に分析を開始するために、2 つの事前定義されたメトリクスグループを提供します。By Risk Classification は、リスクがあるとして Amazon Cognito が識別したリクエストのリスクレベルの詳細を示します。By Request Classification リクエストレベルで集計されたメトリクスを反映します。

集約されたメトリクスグループ	説明
リスク分類別	Amazon Cognito でリスクありとして識別されたリクエスト。
リクエスト分類別	リクエスト別に集約されたメトリクス。

Amazon Cognito ユーザープールのディメンション

Amazon Cognito によってパブリッシュされた使用状況メトリクスの絞り込みには、以下のディメンションが使用されます。ディメンションは、CallCount および ThrottleCount メトリクスのみに適用されます。

ディメンション	説明
Service	リソースを含む AWS サービスの名前。Amazon Cognito の使用状況メトリクスの場合、このディメンションの値は Cognito user pool です。
Type	報告されるエンティティのタイプ。Amazon Cognito の使用状況メトリクスに対する有効な値は API のみです。
Resource	実行中のリソースのタイプ。有効な値はカテゴリ名のみです。
Class	追跡されているリソースのクラス。Amazon Cognito は class ディメンションを使用しません。

CloudWatch コンソールを使用してメトリクスを追跡する

CloudWatch を使用して、Amazon Cognito ユーザープールのメトリクスを追跡して収集することができます。CloudWatch ダッシュボードには、使用するすべての AWS サービスに関するメトリクスが表示されます。CloudWatch を使用してメトリクスアラームを作成できます。アラーム

は、通知を送信する、またはモニタリングしている特定のリソースに変更を行うように設定できます。CloudWatch でサービスクォータメトリクスを表示するには、以下のステップを実行します。

1. [CloudWatch コンソール](#) を開きます。
2. ナビゲーションペインで [Metrics (メトリクス)] を選択してください。
3. [All metrics] (すべてのメトリクス) でメトリクスとディメンションを選択します。
4. メトリクスの横にあるチェックボックスをオンにします。グラフにメトリクスが表示されます。

Note

過去 2 週間に新しいデータポイントがなかったメトリクスは、コンソールに表示されません。また、コンソールの [All metrics] (すべてのメトリクス) タブにある検索ボックスにそれらのメトリクス名またはディメンション名を入力しても表示されず、list-metrics コマンドの結果にも返されません。これらのメトリクスを取得する最適な方法は、AWS CLI で get-metric-data または get-metric-statistics コマンドを使用することです。

クォータの CloudWatch アラームを作成する

Amazon Cognito は、CallCount および ThrottleCount API AWS のサービスクォータに対応する CloudWatch 使用状況メトリクスを提供します。APIs CloudWatch での使用状況の追跡については、「[クォータの使用状況を追跡する](#)」を参照してください。

Service Quotas コンソールで、使用量がサービスクォータに近づいた場合に警告するアラームを作成できます。Service Quotas コンソールを使用して CloudWatch アラームを設定する方法については、「[Service Quotas and CloudWatch alarms](#)」を参照してください。

Service Quotas のメトリクス

Service Quotas を使用して、一元化された場所から Amazon Cognito ユーザープールおよびアイデンティティプールのクォータを表示および管理できます。Service Quotas コンソールを使用して、特定のクォータに関する詳細の表示、クォータ使用率のモニタリング、クォータ引き上げのリクエストを行うことができます。一部のクォータタイプでは、CloudWatch アラームを作成してクォータ使用率を追跡できます。追跡できる Amazon Cognito メトリクスの詳細については、「[クォータの使用状況を追跡する](#)」を参照してください。

Amazon Cognito ユーザープールおよびアイデンティティプールのサービスクォータ使用率を表示するには、次の手順を実行します。

1. [Service Quotas コンソール](#) を開きます。
2. ナビゲーションペインで、[AWS サービス] を選択します。
3. [AWS のサービス] リストで、[Amazon Cognito ユーザープール] または [Amazon Cognito フェデレーテッド ID] を検索して選択します。[Service Quota] (サービスクォータ) ページが表示されます。
4. CloudWatch モニタリングをサポートするクォータを選択します。例えば、Amazon Cognito ユーザープールで [Rate of UserAuthentication requests] を選択します。
5. [Monitoring] (モニタリング) までスクロールダウンします。このセクションは、CloudWatch モニタリングをサポートするクォータに対してのみ表示されます。
6. [Monitoring] (モニタリング) では、現在のサービスクォータ使用率をグラフに表示できます。
7. [Monitoring] (モニタリング) で、1 時間、3 時間、12 時間、1 日、3 日、または 1 週間を選択します。
8. グラフ内の任意の領域を選択して、サービスクォータ使用率の割合を表示します。ここでは、ダッシュボードにグラフを追加する、またはアクションメニューを使用して、CloudWatch コンソールで関連メトリクスを表示する [View in metrics] (メトリクスで表示) を選択することができます。

Amazon Cognito ログイン AWS CloudTrail

Amazon Cognito は AWS CloudTrail、Amazon Cognito のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスであると統合されています。CloudTrail は、Amazon Cognito の API コールのサブセットをイベントとしてキャプチャし、これには Amazon Cognito コンソールと、Amazon Cognito API 操作へのコードコールからのコールが含まれます。証拠を作成する場合、Amazon Cognito のイベントを含めた CloudTrail イベントの Amazon S3 バケットへの継続的な配信を有効にすることができます。追跡を設定しない場合でも、CloudTrail コンソールの [Event history] (イベント履歴) で最新のイベントを表示できます。CloudTrail によって収集された情報を使用して、Amazon Cognito に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエストが行われた日時、および追加の詳細を確認できます。

設定や有効化の方法など、CloudTrail の詳細については、「[AWS CloudTrail ユーザーガイド](#)」を参照してください。

特定の CloudTrail イベントに対する Amazon CloudWatch アラームを作成することもできます。例えば、ID プールの設定が変更された場合にアラームがトリガーされるように CloudWatch を設定できます。詳細については、「[CloudTrail イベントの CloudWatch アラームの作成: 例](#)」を参照してください。

トピック

- [Amazon Cognito が CloudTrail に送信する情報](#)
- [Amazon CloudWatch Logs Insights を使用した Amazon Cognito CloudTrail イベントの分析](#)
- [Amazon Cognito イベントの例](#)

Amazon Cognito が CloudTrail に送信する情報

CloudTrail は、の作成時にオンになります AWS アカウント。Amazon Cognito でサポートされているイベントアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービス イベントとともに CloudTrail イベントに記録されます。AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「[CloudTrail Event 履歴でのイベントの表示](#)」を参照してください。

Amazon Cognito のイベントなど、AWS アカウントのイベントの継続的な記録については、証跡を作成します。CloudTrail の証跡がログファイルを Amazon S3 バケットに配信します。デフォルトでは、コンソールで証跡を作成すると、すべてのリージョンに証跡が適用されます。証跡は、AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをさらに分析して処理するように他の AWS サービスを設定できます。詳細については、以下を参照してください。

- [証跡を作成するための概要](#)
- [CloudTrail がサポートされているサービスと統合](#)
- [amazon SNS の CloudTrail 通知の設定](#)
- [複数のリージョンから CloudTrail ログファイルを受け取る](#)および[複数のアカウントから CloudTrail ログファイルを受け取る](#)

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、「[CloudTrail userIdentity エlement](#)」を参照してください。

の機密データ AWS CloudTrail

ユーザープールと ID プールはユーザーデータを処理するため、Amazon Cognito は CloudTrail イベントの一部のプライベートフィールドを値 `HIDDEN_DUE_TO_SECURITY_REASONS` で非表示にします。Amazon Cognito がイベントに設定しないフィールドの例については、「[Amazon Cognito イベントの例](#)」を参照してください。Amazon Cognito は、パスワードやトークンなど、一般的にユーザー情報を含む一部のフィールドだけを非表示にします。Amazon Cognito は、API リクエスト内の非公開フィールドに入力された個人識別情報の自動検出やマスキングを行いません。

ユーザープールのイベント

Amazon Cognito は、[\[User pool actions\]](#) (ユーザープールのアクション) ページにリストされているすべてのアクションを、イベントとして CloudTrail ログファイルに記録することをサポートします。Amazon Cognito は、ユーザープールイベントを管理イベントとして CloudTrail にログ記録します。

Amazon Cognito ユーザープールの CloudTrail エントリの `eventType` フィールドは、アプリケーションがリクエストした先が [Amazon Cognito ユーザープール API](#) であるのか、[OpenID Connect](#)、[SAML 2.0](#)、または[マネージドログインページのリソースを提供するエンドポイント](#)であるのかを示します。API リクエストには `AwsApiCall` の `eventType` があり、エンドポイントリクエストには `AwsServiceEvent` の `eventType` があります。

Amazon Cognito は、マネージドログインサービスへの以下のリクエストを CloudTrail にイベントとして記録します。

Hosted UI (classic) events

CloudTrail のホストされた UI (クラシック) のイベント

運用	説明
Login_GET , CognitoAuthentication	ユーザーは ログインエンドポイント に認証情報を表示または送信します。
OAuth2_Authorize_GET , Beta_Authorize_GET	ユーザーは 認可エンドポイント を表示します。

運用	説明
OAuth2Response_GET , OAuth2Response_POST	ユーザーは IdP トークンを /oauth2/idpresponse エンドポイントに送信します。
SAML2Response_POST , Beta_SAML2Response_POST	ユーザーは IdP SAML アサーションを /saml2/idpresponse エンドポイントに送信します。
Login_OIDC_SAML_POST	ユーザーは、 ログインエンドポイント でユーザー名を入力し、 IdP identifier と一致させます。
Token_POST , Beta_Token_POST	ユーザーは、認可コードを トークンエンドポイント に送信します。
Signup_GET , Signup_POST	ユーザーは、サインアップ情報を /signup エンドポイントに送信します。
Confirm_GET , Confirm_POST	ユーザーは、ホストされた UI で確認コードを送信します。
ResendCode_POST	ユーザーは、ホストされた UI で確認コードの再送リクエストを送信します。
ForgotPassword_GET , ForgotPassword_POST	ユーザーは、パスワードのリセットリクエストを /forgotPassword エンドポイントに送信します。
ConfirmForgotPassword_GET , ConfirmForgotPassword_POST	ユーザーは、ForgotPassword リクエストを確認するコードを /confirmForgotPassword エンドポイントに送信します。
ResetPassword_GET , ResetPassword_POST	ユーザーは、ホストされた UI で新しいパスワードを送信します。

運用	説明
Mfa_GET, Mfa_POST	ユーザーは、ホストされた UI で多要素認証 (MFA) コードを送信します。
MfaOption_GET , MfaOption_POST	ユーザーは、ホストされた UI で MFA の好みの方法を選択します。
MfaRegister_GET , MfaRegister_POST	ユーザーは、MFA の登録時にホストされた UI で多要素認証 (MFA) コードを送信します。
Logout	ユーザーは、/logout エンドポイントでサインアウトします。
SAML2Logout_POST	ユーザーは、/saml2/logout エンドポイントでサインアウトします。
Error_GET	ユーザーは、ホストされた UI でエラーページを表示します。
UserInfo_GET , UserInfo_POST	ユーザーまたは IdP は、 userInfo エンドポイント と情報を交換します。
Confirm_With_Link_GET	ユーザーは、Amazon Cognito が E メールメッセージで送信したリンクに基づいて確認を送信します。
Event_Feedback_GET	ユーザーが、 脅威保護 イベントに関するフィードバックを Amazon Cognito に送信します。

Managed login events

CloudTrail でのマネージドログインのイベント

運用	説明
login_POST	ユーザーが認証情報を ログインエンドポイント に送信します。
login_continue_POST	既に 1 回サインインしているユーザーが、再度サインインすることを選択します。
forgotPassword_POST	ユーザーがパスワードをリセットします。
selectChallenge_POST	ユーザーが、ユーザー名または認証情報を送信した後で、認証チャレンジに応答します。
confirmUser_GET	ユーザーが 確認または検証 E メールメッセージ でリンクを開きます。
mfa_back_POST	ユーザーが MFA プロンプトの後に [戻る] ボタンを選択します。
mfa_options_POST	ユーザーが MFA オプションを選択します。
mfa_phone_register_POST	ユーザーが電話番号を送信して MFA 要素として登録します。このオペレーションにより、Amazon Cognito は電話番号宛に MFA コードを送信します。
mfa_phone_verify_POST	ユーザーが電話番号で受け取った MFA コードを送信します。
mfa_phone_resendCode_POST	ユーザーが、電話番号に MFA コードを再送信してもらうリクエストを送信します。
mfa_totp_POST	ユーザーが TOTP MFA コードを送信します。
signup_POST	ユーザーが /signup マネージドログインページに情報を送信します。

運用	説明
signup_confirm_POST	ユーザーが E メールまたは SMS メッセージから確認コードを送信します。
verifyCode_POST	ユーザーがパスワードなしの認証のためにワンタイムパスワード (OTP) を送信します。
passkeys_add_POST	ユーザーが新しいパスキー認証情報を登録するためのリクエストを送信します。
passkeys_add_GET	ユーザーがパスキーを登録できるページに移動します。
login_passkey_POST	ユーザーがパスキーでサインインします。

Note

ユーザー固有のリクエストについて、Amazon Cognito は CloudTrail ログに UserSub を記録しますが、UserName は記録しません。ListUsers API を呼び出し、sub のフィルターを使用することで、所定の UserSub のユーザーを見つけることができます。

アイデンティティプールイベント

データイベント

Amazon Cognito は、CloudTrail への Amazon Cognito ID イベントをデータイベントとしてログ記録します。[データイベント](#)は、CloudTrail がデフォルトではログ記録しない大量のデータプレーン API オペレーションです。追加の変更がイベントデータに適用されます。

- [GetCredentialsForIdentity](#)
- [GetId](#)
- [GetOpenIdToken](#)
- [GetOpenIdTokenForDeveloperIdentity](#)
- [UnlinkIdentity](#)

これらの API オペレーションの CloudTrail ログを生成するには、トレイル内のデータイベントを有効にし、Cognito ID プールのイベントセクターを選択する必要があります。詳細については、「AWS CloudTrail ユーザーガイド」の「[証跡のデータイベントの記録](#)」を参照してください。

次の CLI コマンドを使用して、ID プールのイベントセクターをトレイルに追加することもできます。

```
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\" \
      ] \
    } \
  ] \
}"
```

管理イベント

Amazon Cognito は、残りの Amazon Cognito アイデンティティプール API オペレーションを管理イベントとしてログ記録します。CloudTrail は、デフォルトで管理イベント API オペレーションをログ記録します。

Amazon Cognito が CloudTrail にログ記録する Amazon Cognito ID API オペレーションのリストについては、[Amazon Cognito アイデンティティプール API リファレンス](#) をご覧ください。

Amazon Cognito Sync

Amazon Cognito は、すべての Amazon Cognito Sync API オペレーションを管理イベントとしてログ記録します。Amazon Cognito が CloudTrail にログ記録する Amazon Cognito Sync API オペレーションのリストについては、[Amazon Cognito Sync API Reference](#)(Amazon Cognito Sync API リファレンス) をご覧ください。

Amazon CloudWatch Logs Insights を使用した Amazon Cognito CloudTrail イベントの分析

Amazon CloudWatch Logs Insights は、Amazon Cognito CloudTrail イベントをインタラクティブに検索し、分析することを可能にします。イベントを CloudWatch Logs に送信するように証跡を設定すると、CloudTrail は証跡設定に一致するイベントのみを送信します。

Amazon Cognito CloudTrail イベントをクエリまたは調査するには、CloudTrail コンソールで証跡設定で Management events オプションを選択し、AWS リソースで実行される管理オペレーションをモニタリングできるようにします。アカウント内でのエラー、異常なアクティビティ、または異常なユーザー動作を特定したい場合は、オプションで追跡設定の [Insights イベント] オプションを選択することができます。

サンプル Amazon Cognito クエリ

Amazon CloudWatch コンソールでは、以下のクエリを使用できます。

一般的なクエリ

最近追加された 25 件のログイベントを検索します。

```
fields @timestamp, @message | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com"
```

例外を含む、最近追加された 25 件のログイベントのリストを取得します。

```
fields @timestamp, @message | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com" and @message like /Exception/
```

例外とエラーのクエリ

Amazon Cognito ユーザープール sub と共に、最近追加されたエラーコード NotAuthorizedException を伴う 25 件のログイベントを検索します。

```
fields @timestamp, additionalEventData.sub as user | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com" and errorCode=
"NotAuthorizedException"
```

eventName と対応する sourceIpAddress を持つレコードの数を検索します。

```
filter eventSource = "cognito-idp.amazonaws.com"
```

```
| stats count(*) by sourceIPAddress, eventName
```

NotAuthorizedException エラーをトリガーした上位 25 の IP アドレスを検索します。

```
filter eventSource = "cognito-idp.amazonaws.com" and errorCode=
  "NotAuthorizedException"
| stats count(*) as count by sourceIPAddress, eventName
| sort count desc | limit 25
```

ForgotPassword API を呼び出した上位 25 の IP アドレスを検索します。

```
filter eventSource = "cognito-idp.amazonaws.com" and eventName = 'ForgotPassword'
| stats count(*) as count by sourceIPAddress
| sort count desc | limit 25
```

Amazon Cognito イベントの例

Amazon Cognito は、ユーザー認証アクティビティと管理管理アクティビティ AWS CloudTrail に関する情報を に記録します。これは、ユーザープールとアイデンティティプールの両方に適用されます。例えば、同じ証跡で GetId イベントと UpdateIdentityPool イベント、または UpdateAuthEventFeedback イベントと SetRiskConfiguration イベントを表示できます。ユーザープール API のオペレーションに対応していないホストされた UI アクティビティのユーザープールログも表示されます。このセクションでは、表示されるログの例をいくつか示します。オペレーションの CloudTrail イベントスキーマを理解するには、そのオペレーションのリクエストを生成し、証跡に作成するイベントを確認します。

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

トピック

- [ホストされた UI サインアップの CloudTrail イベントの例](#)
- [SAML リクエストの CloudTrail イベントの例](#)
- [トークンエンドポイントへのリクエストの CloudTrail イベントの例](#)
- [CreateIdentityPool の CloudTrail イベントの例](#)
- [GetCredentialsForIdentity の CloudTrail イベントの例](#)

- [GetId の CloudTrail イベントの例](#)
- [GetOpenIdToken の CloudTrail イベントの例](#)
- [GetOpenIdTokenForDeveloperIdentity の CloudTrail イベントの例](#)
- [UnlinkIdentity の CloudTrail イベントの例](#)

ホストされた UI サインアップの CloudTrail イベントの例

以下の CloudTrail イベントの例は、ユーザーがホストされた UI からサインアップしたときに Amazon Cognito がログに記録する情報を示しています。

Amazon Cognito は、新しいユーザーがアプリのサインインページに移動すると、次のイベントをログに記録します。

```
{
  "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 は、新しいユーザーがアプリのサインインページで [Sign up] を選択すると、次のイベントをログに記録します。

```
{
  "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.auth.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_EXAMPLE"
},
"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 は、新しいユーザーがユーザー名を選択し、E メールアドレスを入力し、アプリのサインインページからパスワードを選択すると、次のイベントをログに記録します。Amazon Cognito は、ユーザーの ID に関する識別情報を CloudTrail に記録しません。

```
{
    "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_REASONS"
    ],
    "requiredAttributes[email]":
    [
      "HIDDEN_DUE_TO_SECURITY_REASONS"
    ],
    "response_type":
    [
      "code"
    ],
    "_csrf":
    [
      "HIDDEN_DUE_TO_SECURITY_REASONS"
    ],
    "redirect_uri":
    [
      "https://www.amazon.com"
    ],
    "client_id":
    [
      "1example23456789"
    ],
    "username":
    [
      "HIDDEN_DUE_TO_SECURITY_REASONS"
    ]
  },
  "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
```



```
    "userPoolId": "us-west-2_EXAMPLE"
  },
  "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 は、新規ユーザーがサインアップ後にホストされた UI のユーザー確認ページにアクセスすると、次のイベントをログに記録します。

```
{
  "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.auth.us-west-2.amazoncognito.com",
  "userPoolId": "us-west-2_EXAMPLE"
},
"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 は、ホストされた UI のユーザー確認ページで、Amazon Cognito が送信したコードを E メールメッセージでユーザーが入力したときに、次のイベントをログに記録します。

```

{
  "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_REASONS"
    ],
    "destination":
    [
      "HIDDEN_DUE_TO_SECURITY_REASONS"
    ],
    "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.auth.us-west-2.amazoncognito.com",
"userPoolId": "us-west-2_EXAMPLE"
},
"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"
}
```

SAML リクエストの CloudTrail イベントの例

Amazon Cognito は、SAML IdP で認証されたユーザーが SAML アサーションを `/saml2/idpresponse` エンドポイントに送信すると、次のイベントをログに記録します。

```
{
    "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_REASONS"
    ],
    "SAMLResponse":
    [
      "HIDDEN_DUE_TO_SECURITY_REASONS"
    ]
  },
  "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
  "userPoolId": "us-west-2_EXAMPLE"
},
"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"
}
```

トークンエンドポイントへのリクエストの CloudTrail イベントの例

次の例は、[トークンエンドポイント](#) へのリクエストからのイベントの例です。

Amazon Cognito は、認証して認可コードを受け取ったユーザーがコードを /oauth2/token エンドポイントに送信すると、以下のイベントを記録します。

```
{
  "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.auth.us-west-2.amazoncognito.com",
  "userPoolId": "us-west-2_EXAMPLE"
},
"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"
}
```

バックエンドシステムが /oauth2/token エンドポイントにアクセストークンの client_credentials リクエストを送信すると、Amazon Cognito は次のイベントをログに記録します。

```
{
  "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.auth.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_EXAMPLE"
  },
  "requestID": "4f871256-6825-488a-871b-c2d9f55caff2",
  "eventID": "473e5cbc-a5b3-4578-9ad6-3dfdcb8a6d34",
  "readOnly": false,
  "eventType": "AwsServiceEvent",
  "managementEvent": true,
  "recipientAccountId": "123456789012",
  "serviceEventDetails":
  {
    "serviceAccountId": "111122223333"
  },
  "eventCategory": "Management"
}

```

Amazon Cognito は、アプリが更新トークンを新しい ID に交換し、/oauth2/token エンドポイントとトークンにアクセスすると、次のイベントをログに記録します。

```

{
  "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.auth.us-west-2.amazoncognito.com",
    "userPoolId": "us-west-2_EXAMPLE"
},
"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"
}

```

CreateIdentityPool の CloudTrail イベントの例

次の例は、CreateIdentityPool アクションに対するリクエストのログエントリを示しています。リクエストは Alice という IAM ユーザーによって行われました。

```

{
    "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"
  }
}

```

GetCredentialsForIdentity の CloudTrail イベントの例

次の例は、GetCredentialsForIdentity アクションに対するリクエストのログエントリを示しています。

```

{
  "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_aaaaaa":
"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"
}

```

GetId の CloudTrail イベントの例

次の例は、GetId アクションに対するリクエストのログエントリを示しています。

```

{
    "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_aaaaaaaaa":
"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"
}

```

GetOpenIdToken の CloudTrail イベントの例

次の例は、GetOpenIdToken アクションに対するリクエストのログエントリを示しています。

```

{
    "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_aaaaaaa":
"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"
}

```

GetOpenIdTokenForDeveloperIdentity の CloudTrail イベントの例

次の例は、GetOpenIdTokenForDeveloperIdentity アクションに対するリクエストのログエントリを示しています。

```

{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIEXAMPLE:johns-AssumedRoleSession",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/johns-AssumedRoleSession",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",

```

```

        "principalId": "AROAIEXAMPLE",
        "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_REASONS"
    }
},
"responseElements": {
    "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"
},
"requestID": "b807df87-57e7-4dd6-b90c-b06f46a61c21",
"eventID": "f26fed91-3340-4d70-91ae-cdf555547b76",
"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"
}

```

UnlinkIdentity の CloudTrail イベントの例

次の例は、UnlinkIdentity アクションに対するリクエストのログエントリを示しています。

```
{
  "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-1_aaaaaaaa":
"HIDDEN_DUE_TO_SECURITY_REASONS"
    },
    "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",
    "loginsToRemove": ["cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaa"]
  },
  "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"
}
```

インターフェイスエンドポイント (AWS PrivateLink) を使用して Amazon Cognito にアクセスする

を使用して AWS PrivateLink、VPC と Amazon Cognito の間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、または Direct Connect 接続を使用せずに、VPC 内にあるかのように Amazon Cognito にアクセスできます。VPC 内のインスタンスは、Amazon Cognito にアクセスするためにパブリック IP アドレスを必要としません。

このプライベート接続を確立するには、AWS PrivateLinkを利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、Amazon Cognito 宛てのトラフィックのエントリポイントとして機能するリクエスト管理のネットワークインターフェイスです。

詳細については「AWS PrivateLink ガイド」の「[Access AWS のサービス through AWS PrivateLink](#)」を参照してください。

Important

以下の認証タイプは、現在 ではサポートされていません AWS PrivateLink。

1. OAuth 2.0 クライアント認証情報フローによるマシンツーマシン (M2M) 認可
2. マネージドログインとクラシックホスト UI を使用してサインインします。

トピック

- [AWS PrivateLink 統合の認証フロー](#)
- [のオペレーションモード AWS PrivateLink](#)
- [Amazon Cognito に関する考慮事項](#)
- [リソースコントロールポリシーによるアクセスの制御](#)
- [Amazon Cognito のインターフェイスエンドポイントを作成する](#)
- [インターフェイスエンドポイントのエンドポイントポリシーを作成する](#)
- [AWS PrivateLink オペレーションのアイデンティティベースのポリシーを作成する](#)

AWS PrivateLink 統合の認証フロー

次の表は、VPCs 内のクライアントが使用できる認証フローと、それらを管理するために適用できる IAM ポリシーを示しています。ユーザープールへのリクエストで評価できるポリシーは、リソースコントロールポリシー (RCPs)、VPC エンドポイントポリシー、およびアイデンティティベースのポリシーです。

[リソース]	認証フロー	クライアントが VPC エンドポイントを転送したときに評価されるポリシー	クライアントオリジンがパブリックの場合に評価されるポリシー
ユーザープール	マネージドログインとクラシックホスト UI サインイン	なし (アクセスなし) ¹	なし ²
ユーザープール	Machine-to-machine認可	なし (アクセスなし) ¹	なし ²
ユーザープール	SDK および REST API の認証されていないリクエスト	RCPs、VPC エンドポイントポリシー ³	RCPs
ユーザープール	SDK および REST API SigV4 認証リクエスト	RCPs、VPC エンドポイントポリシー、アイデンティティベースのポリシー ³	RCPs、アイデンティティベースのポリシー
ID プール	SDK および REST API の認証されていないリクエスト (基本フロー と 拡張フロー)	RCPs、VPC エンドポイントポリシー	RCPs
ID プール	SDK および REST API SigV4 認証リクエスト (開発者認証フロー)	RCPs、アイデンティティベースのポリシー	RCPs、アイデンティティベースのポリシー

¹ VPC エンドポイントは、ユーザープールドメインのリクエストを受け入れません。クライアントにインターネットへのルートがある場合、NAT が適用され、オリジンがパブリックになります。

² ユーザープールドメインが存在すると、VPC エンドポイントを通過するユーザープールリクエストが完了できなくなります。すべてのクライアントは、ユーザープールドメインと API サービスエンドポイントにのみパブリックトランジットパスを取ることができ、VPC エンドポイントをユーザープールで使用できなくなります。ドメインが割り当てられているユーザープールは と互換性がありません AWS PrivateLink。

³ ユーザープールにはドメインを割り当てることはできません。

のオペレーションモード AWS PrivateLink

次の実装モデルは、AWS PrivateLink および Amazon Cognito でサポートされています。

[リソース]	実装	アクション
ユーザープール	完全プライベート SDK または REST API アプリケーション	1. ドメインの削除 2. VPC エンドポイントを作成する 3. VPC を除く Deny すべての cognito-idp アクションに RCP を設定する
ユーザープール	パブリックおよびプライベート IP	1. ドメインの削除 2. VPC エンドポイントを作成する
ユーザープール	プライベートまたはパブリック OAuth 2.0 認可サーバー	1. VPC では使用できません
ID プール	完全プライベート	1. VPC エンドポイントを作成する 2. VPC を除く Deny すべての cognito-identity アクションに RCP を設定する
ID プール	パブリックおよびプライベート IP	1. VPC エンドポイントを作成する

Amazon Cognito に関する考慮事項

Amazon Cognito のインターフェイスエンドポイントを設定する前に、「AWS PrivateLink ガイド」の「[考慮事項](#)」を参照してください。Amazon Cognito は、インターフェイスエンドポイントを介したすべての Amazon Cognito API アクションの呼び出しをサポートしています。これらのオペ

レーションの詳細については、[Amazon Cognito ユーザープール API リファレンス](#) および [Amazon Cognito フェデレーティッドアイデンティティ API リファレンス](#) を参照してください。

AWS PrivateLink for Amazon Cognito は商用 AWS リージョンでのみ使用できます。

トピック

- [ユーザープールと AWS PrivateLink](#)
- [ID プールと AWS PrivateLink](#)

ユーザープールと AWS PrivateLink

インターフェイスエンドポイントを介してすべてのユーザープール API オペレーションにリクエストを行うことができますが、クライアント認証情報の付与やマネージドログインなど、アプリケーションがユーザープール OAuth 2.0 認可サーバーにリクエストするオペレーションにはリクエストできません。

cognito-idp ユーザープール API には、[認証されていない、認証された、トークンが認可された API オペレーション](#)があります。VPC エンドポイントとリソースコントロールポリシーで認証されたオペレーションのアクセス許可を付与できます。ID ベースのポリシーとは異なり、認証されていないオペレーションやトークン認証されたオペレーションのアクセス許可を付与することもできます。VPC エンドポイントとリソースコントロールポリシータイプは、それ以外のパブリックオペレーションのリクエストを評価、拒否、または許可できます。

ドメインエンドポイントへのリクエストもパブリックですが、ポリシーで評価することはできません。VPC プライベート DNS は、ユーザープールドメインのリクエストを VPC エンドポイントにルーティングしません。ドメインサービスのリクエストは、パブリックインターネットパスを介してのみ行うことができます。詳細については、「[ユーザープールオペレーションに対するポリシーの影響](#)」を参照してください。

トピック

- [サポートされているオペレーション](#)
- [ユーザープールオペレーションに対するポリシーの影響](#)

サポートされているオペレーション

VPC 内のシステムは、[ユーザープール API アクション](#)にリクエストを送信できますが、ユーザープール [ドメインエンドポイント](#)には送信できません。マシンツーマシン (M2M)、フェデレーティッド

ドサインイン、[認可コード付与](#)などのドメインエンドポイントを使用する OpenID Connect (OIDC) および OAuth 2.0 ワークフローは、VPC エンドポイントからはアクセスできません。 [machine-to-machine ???](#) VPC エンドポイントポリシーは、これらの HTTP ワークフローには影響せず、処理できません。VPC 内からのドメインエンドポイントへのリクエストは、常にインターフェイスエンドポイントで失敗しますが、ユーザープールの VPC エンドポイントを設定すると、パブリック DNS とルーティングを通じて引き続き使用できます。

VPC 内のシステムからドメインが割り当てられないように、Amazon Cognito はインターフェイスエンドポイントでの `CreateUserPoolDomain` リクエストをブロックします。これにより、VPC にあるシステムからユーザープールにドメインを追加できなくなります。すべてのシステムからドメインが追加されないようにするには、次の例のような [リソースコントロールポリシー](#) (RCP) を適用します AWS アカウント。このポリシーは、指定されたユーザープールに対する `CreateUserPoolDomain` アクションをブロックします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Deny",
      "Action": [
        "cognito-idp:CreateUserPoolDomain"
      ],
      "Resource": "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-east-1_EXAMPLE"
    }
  ]
}
```

ユーザープールにはドメインがあり、いずれの場合もそのドメインを使用することはできません AWS PrivateLink。 `cognito-idp` [サービスエンドポイント](#) へのすべての SDK ベースの [ユーザープール API リクエスト](#) は AWS PrivateLink、を除く を通じてリクエストを受け入れます `CreateUserPoolDomain`。ユーザープール API サービスエンドポイントとドメインエンドポイントは、パブリックインターネットパスを介して常にアクセスできます。パブリックソースからのアクセスに対処するには、[AWS WAF ウェブ ACLs](#)。

ユーザープールオペレーションに対するポリシーの影響

すべてのユーザープール API オペレーションは、通常はパブリックおよび未認証であっても、VPC エンドポイントポリシーとリソースコントロールポリシー (RCPs) で制御できます。VPC 条件キー

を使用して、アイデンティティベースのポリシーのユーザープールアクセスに制限を適用することもできます。[SigV4 形式の](#)認証情報を含むリクエストのみが、アイデンティティベースのポリシーで制御できます。マネージドログインとクラシックホスト UI オペレーションは別のカテゴリであり、VPC トランジットやアクションへのあらゆる種類のポリシーの適用の対象にはなりません。

認証されていないオペレーション

クライアント側のアプリケーションの Amazon Cognito オペレーションは SigV4 で認証されません。オペレーションの例は、のポリシーの例にあります[インターフェイスエンドポイントのエンドポイントポリシーを作成する](#)。認証されていないオペレーションのその他の例は、GetUserおよびAssociateSoftwareToken。これらのオペレーションを[アイデンティティベースのポリシー](#)に追加しても、効果はありません。ただし、VPC エンドポイントポリシーと [RCPs](#) では、これらのアクションへのアクセスを許可または制限できます。

認証されていないオペレーションは IAM プリンシパルに関連付けられません。VPC エンドポイントポリシーまたは RCP は、これらのアクションのすべてのプリンシパルを許可する必要があります。

認証されたオペレーション

ユーザープール管理とサーバー側の認証の API オペレーションは、SigV4 で認証されます。認証されたオペレーションの場合、VPC [エンドポイントに適用するエンドポイントポリシー](#)、組織内の[リソースコントロールポリシー](#)、およびプリンシパルに適用する[アイデンティティベースのポリシー](#)を使用してプリンシパルを制限できます。ID ベースポリシーとリソースコントロールポリシーは、aws:SourceVpc や などの[ネットワークベースの条件キー](#)を使用して VPC 対応ですaws:SourceVpce。

ユーザープールの API オペレーションのサーバー側、クライアント側、管理クラスの詳細については、「」を参照してください[API 認証と SDK 認証の認可モデル](#)。

ID プールと AWS PrivateLink

Amazon Cognito ID プールは、を介したすべての API オペレーションをサポートします AWS PrivateLink。

トピック

- [サポートされているオペレーション](#)
- [AWS STS 統合によるネットワークコンテキストの制限](#)
- [サービス固有のコンテキストキー](#)

サポートされているオペレーション

すべての ID プール API オペレーションは、インターフェイスエンドポイントを介してサポートされます。ID プールにはドメインエンドポイントがなく、同じ制限の対象ではありません。ただし、ID プールには、との統合により、ネットワークベースのアクセスコントロールに関する特定の考慮事項があります AWS STS。

AWS STS 統合によるネットワークコンテキストの制限

ID プールは オペレーションを使用して AWS STS AssumeRoleWithWebIdentity 一時的な AWS 認証情報を提供します。拡張認証フロー AWS PrivateLink で ID プールが AWS STS を呼び出す場合、aws:SourceIp、などのネットワークコンテキストキーにはaws:SourceVpc、アプリケーションのネットワークコンテキストからではなく、ID プールサービスインフラストラクチャの値aws:SourceVpcが含まれます。

IAM ロールの信頼ポリシーまたはリソースコントロールポリシー (RCPs) がネットワークベースの条件キーを使用してアクセスを制限すると、ID プールオペレーションが予期せず拒否される可能性があります。この制限に対処するには、次のいずれかの方法を使用できます。

サービス識別のプリンシパルタグ

ID プールで使用される IAM ロールにタグを付け、プリンシパルに適切なタグがある場合にオペレーションを許可するようにポリシーを変更します。まず、ID プールロールにタグを追加します。

```
aws iam tag-role \
  -\--role-name MyIdentityPoolRole \
  -\--tags Key=CognitoServiceCall,Value=true
```

次に、ネットワークベースのポリシーを変更して、タグ付けされたプリンシパルを許可します。例えば、RCP の場合:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Resource": "*",
      "Condition": {
```

```

        "NotIpAddress": {
            "aws:SourceIp": ["allowed-ip-ranges"]
        },
        "StringNotEqualsIfExists": {
            "aws:ResourceTag/CognitoServiceCall": "true"
        }
    }
}
]
}

```

サービス固有のコンテキストキー

ID プールは、VPC エンドポイントポリシーと RCPs。これらのコンテキストキーを使用すると、きめ細かなアクセスコントロールを有効にし、ポリシーで認証されたユーザーと認証されていないユーザーを区別できます。

[GetId](#)、[GetCredentialsForIdentity](#)[GetOpenIdToken](#)、[UnlinkIdentity](#) などのnon-SigV4 オペレーションで利用可能なサービス固有のコンテキストキー

- `cognito-identity-unauth:IdentityPoolArn` - 認証されていないユーザーの ID プール ARN でアクセスをフィルタリングします
- `cognito-identity-unauth:AccountId` - 認証されていないユーザーの AWS アカウント ID でアクセスをフィルタリングします
- `cognito-identity-auth:IdentityPoolArn` - 認証されたユーザーの ID プール ARN でアクセスをフィルタリングします
- `cognito-identity-auth:AccountId` - 認証されたユーザーの AWS アカウント ID でアクセスをフィルタリングします

[DeleteIdentities](#) や [DescribeIdentity](#) などの SigV4 オペレーションで利用できるサービス固有のコンテキストキー

- `cognito-identity:IdentityPoolArn` - ID プール ARN でアクセスをフィルタリングします

次の例に示すように、VPC エンドポイントポリシーでこれらのコンテキストキーを使用して、認証ステータスに基づいてアクセスを制限できます。

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": "*",
    "Action": [
      "cognito-identity:GetId",
      "cognito-identity:GetCredentialsForIdentity"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "cognito-identity-unauth:IdentityPoolArn": "arn:aws:cognito-identity:us-east-1:123456789012:identitypool/us-east-1:12345678-ffff-ffff-ffff-123456"
      }
    }
  }
]
}

```

リソースコントロールポリシーによるアクセスの制御

Amazon Cognito は、リソースコントロール[ポリシー \(RCP\) によるリソース](#)へのアクセスの制御をサポートしています。RCPs [ネットワークベースの条件キー](#)を使用すると、RCPs はユーザープールと ID プール AWS PrivateLink へのアクセスを許可するネットワークとアクションを定義できます。RCPs の Action ステートメントは、認証されたユーザープール API オペレーションと認証されていないユーザープール API オペレーションの両方へのアクセスを制御できます。

たとえば、次のポリシー例では、特定の VPC からすべてのユーザープールへのアクセスを禁止しています。

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCognitoAccessOutsideVPC",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "cognito-idp:*",
      "Resource": "*",
      "Condition": {
        "StringNotEqualsIfExists": {

```



```
        "aws:SourceVpc": "vpc-02d6770f46ef1653b"
    }
}
}
]
}
```

Amazon Cognito のインターフェイスエンドポイントを作成する

Amazon Cognito のインターフェイスエンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface () を使用して作成できますAWS CLI。詳細については、「AWS PrivateLink ガイド」の「[インターフェイスエンドポイントを作成](#)」を参照してください。

次のサービス名を使用して、Amazon Cognito ユーザープールのインターフェイスエンドポイントを作成します。

```
com.amazonaws.region.cognito-idp
```

次のサービス名を使用して、Amazon Cognito ID プールのインターフェイスエンドポイントを作成します。

```
com.amazonaws.region.cognito-identity
```

インターフェイスエンドポイントのプライベート DNS を有効にすると、デフォルトのリージョン DNS 名を使用して Amazon Cognito に API リクエストを行うことができます。たとえば、ユーザープールcognito-idp.us-east-1.amazonaws.comの場合は、アイデンティティプールcognito-identity.us-east-1.amazonaws.comの場合は です。

インターフェイスエンドポイントのエンドポイントポリシーを作成する

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーでは、インターフェイスエンドポイントを介して Amazon Cognito へのフルアクセスを許可します。VPC から Amazon Cognito に許可されるアクセスを制御するには、インターフェイスエンドポイントにカスタムエンドポイントポリシーをアタッチします。

エンドポイントポリシーは以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、IAM ユーザー、IAM ロール)。

- 実行可能なアクション。
- このアクションを実行できるリソース。
- リクエストが許可または拒否される前に満たす必要がある条件。

詳細については、「AWS PrivateLink ガイド」の「[Control access to services using endpoint policies](#)」を参照してください。

例: ユーザープールアクションの VPC エンドポイントポリシー

ユーザープールのカスタムエンドポイントポリシーの例を次に示します。このポリシーをインターフェイスエンドポイントにアタッチすると、すべてのリソースのすべてのプリンシパルに対して、リストされたユーザープールアクションへのアクセスが許可されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:assumed-role/MyWebAppRole/MyWebAppSession"
      },
      "Effect": "Allow",
      "Action": [
        "cognito-idp:AdminInitiateAuth",
        "cognito-idp:AdminRespondToAuthChallenge",
        "cognito-idp:AdminSetUserPassword"
      ],
      "Resource": "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-east-1_EXAMPLE"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cognito-idp:InitiateAuth",
        "cognito-idp:RespondToAuthChallenge",
        "cognito-idp:ForgotPassword",
        "cognito-idp:ConfirmForgotPassword"
      ],
      "Resource": "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-east-1_EXAMPLE"
    }
  ]
}
```

```
}
```

例: ID プールアクションの VPC エンドポイントポリシー

ID プールのカスタムエンドポイントポリシーの例を次に示します。このポリシーは、サービス固有のコンテキストキーを使用して、特定の ID プールから認証されたユーザーへのアクセスを制限します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "cognito-identity:GetId",
        "cognito-identity:GetCredentialsForIdentity",
        "cognito-identity:GetOpenIdToken"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cognito-identity-auth:IdentityPoolArn": "arn:aws:cognito-identity:us-east-1:123456789012:identitypool/us-east-1:12345678-ffff-ffff-ffff-123456"
        }
      }
    }
  ]
}
```

AWS PrivateLink オペレーションのアイデンティティベースのポリシーを作成する

[ID ベースのポリシー](#)は、AWS プリンシパルにアタッチできる IAM リソースです。IAM 認証オペレーションのアイデンティティベースのポリシーを使用して、VPC エンドポイントを介して Amazon Cognito へのアクセスを制御できます。エンドポイントポリシーとは異なり、アイデンティティベースのポリシーでは、認証されていないオペレーションのアクセス許可を設定することはできません。認証または管理オペレーションには、[署名バージョン 4](#) の承認が必要です。ユーザープールの場合、認証されたオペレーションには、[AdminInitiateAuth](#) などのサーバー側の認証リクエストと [UpdateUserPool](#) などの管理リクエストが含まれます。ID プールの場合、認証されたオペレーションには [DeleteIdentities](#) や [DescribeIdentity](#) などの管理リクエストが含まれます。

アイデンティティベースのポリシーでは、次の情報を指定します。

- 実行可能なアクション。
- このアクションを実行できるリソース。
- リクエストが許可または拒否される前に満たす必要がある条件。

例: ユーザープールのサーバー側の認証のためのアイデンティティベースのポリシー

次のポリシー例では、指定されたエンドポイントから、指定されたユーザープール内のリストされたユーザープールアクションへのアクセスを許可します。このポリシーをウェブアプリケーションの引き受けた IAM ロールに適用します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cognito-idp:AdminInitiateAuth",
        "cognito-idp:AdminRespondToAuthChallenge",
        "cognito-idp:AdminSetUserPassword"
      ],
      "Resource": "arn:aws:cognito-idp:us-east-1:123456789012:userpool/us-east-1_EXAMPLE",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpce": "vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

例: ID プール管理オペレーションの ID ベースのポリシー

次のポリシー例では、指定された VPC エンドポイントから ID プール管理アクションへのアクセスを許可します。ID プール管理を実行する必要がある IAM プリンシパルにこのポリシーを適用します。

```
{
  "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "cognito-identity:DeleteIdentities",
      "cognito-identity:DescribeIdentity"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:SourceVpce": "vpce-1a2b3c4d"
      },
      "StringEquals": {
        "cognito-identity:IdentityPoolArn": "arn:aws:cognito-identity:us-east-1:123456789012:identitypool/us-east-1:12345678-ffff-ffff-ffff-123456"
      }
    }
  }
]
```

Amazon Cognito のコンプライアンス検証

サードパーティーの監査者は、複数のコンプライアンスプログラムの一環として Amazon Cognito のセキュリティと AWS コンプライアンスを評価します。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

特定のコンプライアンスプログラム AWS の範囲内のサービスのリストについては、「[コンプライアンスAWS プログラムの範囲内のサービスコンプライアンス](#)」を参照してください。一般的な情報については、「[AWS コンプライアンスプログラム](#)」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「[Downloading reports in AWS Artifact](#)」を参照してください。

Amazon Cognito の使用時におけるユーザーのコンプライアンス責任は、データの機密性、会社のコンプライアンス目的、および適用される法律と規制に応じて異なります。AWS は、コンプライアンスに役立つ以下のリソースを提供しています。

- 「[セキュリティ & コンプライアンス クイックリファレンスガイド](#)」 – これらのデプロイガイドには、アーキテクチャ上の考慮事項の説明と、AWSでセキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイするためのステップが記載されています。

- [HIPAA セキュリティとコンプライアンスのためのアーキテクチャホワイトペーパー](#) – このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- [AWS コンプライアンスリソース](#) – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- [「デベロッパーガイド」のルールを使用してリソースを評価する](#) – AWS Config は、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。
AWS Config
- [AWS Security Hub CSPM](#) – この AWS サービスは、内のセキュリティ状態を包括的に把握 AWS し、セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立ちます。

Amazon Cognito の耐障害性

AWS グローバルインフラストラクチャは、AWS リージョンとアベイラビリティーゾーンを中心に構築されています。リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、フォールトトレランス、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾンの詳細については、[AWS 「グローバルインフラストラクチャ」](#)を参照してください。

トピック

- [リージョンデータに関する考慮事項](#)

リージョンデータに関する考慮事項

Amazon Cognito ユーザープールは、それぞれ 1 つの AWS リージョンで作成され、そのリージョンにのみユーザープロファイルデータを保存します。ユーザープールは、オプション機能の設定方法に応じて、ユーザーデータを別の AWS リージョンに送信できます。

- デフォルトのno-reply@verificationemail.com E メールアドレス設定は Amazon Cognito ユーザープールでの E メールアドレスのルーティング検証に使用されます。E メールは関連付けられているユーザープールと同じリージョン経由でルーティングされます。

- Amazon Cognito ユーザープールで Amazon Simple Email Service (Amazon SES) を設定するために別の E メールアドレスが使用されている場合、その E メールアドレスは Amazon SES の E メールアドレスに関連付けられた AWS リージョンを介してルーティングされます。
- Amazon Cognito ユーザープールからの SMS メッセージは、「[E メールまたは電話による検証の設定](#)」で特に指定されていない限り、同じリージョンの Amazon SNS 経由で送信されます。
- Amazon Cognito ユーザープールで Amazon Pinpoint 分析が使用される場合、イベントデータは米国東部 (バージニア北部) リージョンにルーティングされます。

Note

Amazon Pinpoint は、北米、欧州、アジア、オセアニアの複数の AWS リージョンで利用できます。Amazon Pinpoint リージョンには、Amazon Pinpoint API が含まれています。Amazon Pinpoint リージョンが Amazon Cognito でサポートされている場合、Amazon Cognito は同じ Amazon Pinpoint リージョン内の Amazon Pinpoint プロジェクトにイベントを送信します。リージョンが Amazon Pinpoint でサポートされていない場合、Amazon Cognito は us-east-1 でのイベントの送信のみをサポートします。Amazon Pinpoint の詳細なリージョン情報については、「[Amazon Pinpoint エンドポイントとクォータ](#)」および「[Amazon Cognito ユーザープールを使用して Amazon Pinpoint 分析を使用する](#)」を参照してください。

Amazon Cognito のインフラストラクチャセキュリティ

マネージドサービスである Amazon Cognito は、AWS グローバルネットワークセキュリティで保護されています。AWS セキュリティサービスと [ガインフラストラクチャ AWS](#) を保護する方法については、[AWS 「クラウドセキュリティ」](#) を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「[Infrastructure Protection](#)」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で Amazon Cognito にアクセスします。クライアントは次をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

Amazon Cognito ユーザープールの設定と脆弱性分析

AWS は、ゲストオペレーティングシステム (OS) やデータベースのパッチ適用、ファイアウォール設定、ディザスタリカバリなどの基本的なセキュリティタスクを処理します。これらの手順は適切な第三者によって確認され、証明されています。詳細については、以下のリソースを参照してください。

- [Amazon Cognito のコンプライアンス検証](#)
- [責任共有モデル](#)

AWS Amazon Cognito の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを記述するよりも AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する [IAM カスタマー マネージドポリシーを作成する](#) には時間と専門知識が必要です。すぐに開始するには、AWS マネージドポリシーを使用できます。これらのポリシーは一般的なユースケースを対象としており、AWS アカウントで利用できます。AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「[AWS 管理ポリシー](#)」を参照してください。

AWS サービスは、AWS 管理ポリシーを維持および更新します。AWS 管理ポリシーのアクセス許可は変更できません。サービスでは新しい機能を利用できるようにするために、AWS マネージドポリシーに権限が追加されることがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS 管理ポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が損なわれることはありません。

さらに、は、複数のサービスにまたがるジョブ関数の マネージドポリシー AWS をサポートしています。例えば、ReadOnlyAccess AWS 管理ポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新機能を起動すると、は新しいオペレーションとリソースの読み取り専用アクセス許可 AWS を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「[AWS のジョブ機能のマネージドポリシー](#)」を参照してください。

AWS Amazon Cognito へのアクセスを許可する マネージド IAM ポリシー

- AmazonCognitoPowerUser - ID プールとユーザープールのあらゆる側面へのアクセスと管理のための許可。このポリシーの許可を確認するには、「[AmazonCognitoPowerUser](#)」を参照してください。
- AmazonCognitoReadOnly - ID プールとユーザープールへの読み取り専用アクセスのための許可。このポリシーの許可を確認するには、「[AmazonCognitoReadOnly](#)」を参照してください。
- AmazonCognitoDeveloperAuthenticatedIdentities - 認証システムが Amazon Cognito と統合するための許可。このポリシーの許可を確認するには、「[AmazonCognitoDeveloperAuthenticatedIdentities](#)」を参照してください。

これらのポリシーは Amazon Cognito チームによって維持されるため、新しい API が追加されても、ユーザーは引き続き同じアクセスレベルを維持することができます。

Note

新しいアイデンティティプールを作成すると、認証済みユーザーアクセスとゲストユーザーアクセス用の新しいロールを自動的に作成できます。新しい IAM ロールでアイデンティティプールを作成する管理者には、ロールを作成するための IAM アクセス許可も必要です。

認証されていないゲストアクセスを持つ ID プールは、認証されていないユーザーに[セッション](#)ポリシーとして追加の AWS 管理ポリシーを適用します。この AWS 管理ポリシーには、意図した管理用途はありません。代わりに、アイデンティティプールの[拡張認証フロー](#)でゲストユーザーに適用できるアクセス許可の範囲が制限されます。詳細については、「[IAM ロール](#)」を参照してください。

AWS Amazon Cognito がゲストユーザーに付与する マネージド IAM ポリシー

- AmazonCognitoUnAuthedIdentitiesSessionPolicy - インラインセッションポリシーと組み合わせて、IAM 管理者がアイデンティティプールゲストユーザーに付与できるアクセス許可を制限します。Amazon Cognito は、このポリシーをゲストセッションに自動的に適用します。詳細については、「[ゲストの AWS マネージドセッションポリシー](#)」を参照してください。

Amazon Cognito による AWS マネージドポリシーの更新

このサービスがこれらの変更の追跡を開始してからの Amazon Cognito の AWS マネージドポリシーの更新に関する詳細を表示します。このページへの変更に関する自動アラートについては、Amazon Cognito の「[ドキュメント履歴](#)」ページで RSS フィードにサブスクライブしてください。

変更	説明	日付
AmazonCognitoPowerUser - 変更	Amazon Cognito は、Amazon Cognito ユーザープールの管理パワーユーザーの AWS End User Messaging SMS API オペレーション DescribeAccountAttributes の使用を許可する新しいアクションを追加しました。	2025 年 2 月 27 日
AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy - 変更	Amazon Cognito は、ID プール内の認証されていない (ゲスト) ユーザー AWS Key Management Service に対しての使用を許可する新しいアクションを追加しました。	2024 年 10 月 30 日
AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy - 変更	Amazon Cognito は、アイデンティティプール内の認証されていない (ゲスト) ユーザーに対して Amazon Location Service の使用を許可する新しいアクションを追加しました。	2024 年 8 月 9 日
AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy - 新しいポリシー	ID プール内のゲストユーザーの特権スコープダウンのための AWS マネージドポリシーを追加しました。	2023 年 7 月 14 日

変更	説明	日付
AmazonCognitoPowerUser および AmazonCognitoReadOnly – 変更	パワーユーザーが Amazon Cognito ユーザープールへの AWS WAF ウェブ ACLs の関連付けを表示および管理できるようにする新しいアクセス許可を追加しました。 読み取り専用ユーザーが Amazon Cognito ユーザープールへの AWS WAF ウェブ ACLs の関連付けを表示できるようにする新しいアクセス許可を追加しました。	2022 年 7 月 19 日
AmazonCognitoPowerUser - 変更	Amazon Cognito による Amazon Simple Email Service PutIdentityPolicy および ListConfigurationSets のオペレーションの呼び出しを可能にする新しい許可が追加されました。 この変更により、Amazon Cognito ユーザープールは、Amazon SES 送信認可ポリシーを更新し、ユーザープールの E メール送信設定で Amazon SES の設定を適用できるようになります。	2021 年 11 月 17 日

変更	説明	日付
AmazonCognitoPower User - 変更	Amazon Cognito による Amazon Simple Notification Service の GetSMSSandboxAccountStatus オペレーションの呼び出しを可能にする新しい許可が追加されました。 この変更により、Amazon Cognito ユーザープールは、ユーザープールを通じてすべてのエンドユーザーにメッセージを送信するために、Amazon Simple Notification Service のサンドボックスから乗り換える必要があるかどうかを判断できるようになります。	2021 年 6 月 1 日
Amazon Cognito が変更の追跡を開始	Amazon Cognito は、AWS 管理ポリシーの変更の追跡を開始しました。	2021 年 3 月 1 日

Amazon Cognito のトラブルシューティング

この章では、Amazon Cognito の使用中に発生する可能性がある一般的な問題の解決策を示します。Amazon Cognito の実装は、認証フロー、ユーザープール設定、ID フェデレーション設定のすべてにわたって、さまざまな課題に直面する可能性があります。新しいアプリケーションを開発する場合でも、既存のアプリケーションを維持する場合でも、このトラブルシューティングガイドは一般的な問題をすばやく特定して解決するのに役立ちます。

カスタムドメインの設定エラー

Amazon Cognito でカスタムドメイン名を設定する場合、エラーメッセージが表示されることがあります。一般的なエラーには、検証の問題、証明書の問題、ドメインの競合などがあります。

Custom domain is not a valid subdomain

このエラーは、親ドメインの DNS 解決の問題を示しています。Amazon Cognito は最上位ドメインをサポートしていないため、親ドメインには検証のための DNS A レコードが必要です。

問題

このエラーは、親ドメインの DNS 解決の問題を示しています。Amazon Cognito は最上位ドメインをサポートしていないため、親ドメインには検証のための DNS A レコードが必要です。

ソリューション

- 親ドメインの A レコードを作成する: カスタムドメインの親ドメインで DNS 設定に A レコードを作成する必要があります。
- 例: カスタムドメインが `auth.xyz.yourdomain.com` である場合、親ドメインは `xyz.yourdomain.com` です。 `xyz.yourdomain.com` をカスタムドメインとして設定する場合、親ドメインは `yourdomain.com` です。
- 親ドメインは、有効な IP アドレスに解決する必要があります。実際の IP アドレスを指さない場合は、ダミーの IP アドレス (8.8.8.8 など) を使用できます。
- DNS 伝達の検証 (オプション、ただし推奨): DNS プロバイダーが変更を伝達したことを確認するには、`dig` コマンドを実行できます。
 - `auth.xyz.yourdomain.com` をカスタムドメインとして使用する場合: `dig A xyz.yourdomain.com +short`
 - `xyz.yourdomain.com` をカスタムドメインとして使用する場合: `dig A yourdomain.com +short`

- コマンドは、設定した IP アドレスを返す必要があります。そうでない場合は、変更が完全に反映されるまで待ちます。
- 親ドメインの A レコードを削除する: Amazon Cognito でカスタムドメインを正常に作成したら、親ドメイン用に作成した A レコード (ダミーである場合) を削除できます。

詳細については、「[ホストされた UI で独自のドメインを使用する](#)」を参照してください。

Domain already associated with another user pool

カスタムドメイン名は、すべての AWS アカウント および リージョンで一意である必要があります。

問題

カスタムドメイン名は、すべての AWS アカウント および リージョンで一意である必要があります。

ソリューション

- 新しいユーザープールにドメイン名を使用するには、現在関連付けられているユーザープールからカスタムドメインを削除する必要があります。
- 削除後に待機する: カスタムドメインが最初のユーザープールから完全に削除されるまで時間がかかります。削除直後に同じ名前で作成すると、このエラーが発生する可能性があります。数分待ってから、もう一度試してください。

One or more of the CNAMEs that you provided are already associated with a different resource

カスタムドメインを作成すると、Amazon Cognito は AWS マネージド Amazon CloudFront ディストリビューションを作成します。ドメイン名は、1 つの Amazon CloudFront ディストリビューションでのみ使用できます。このエラーは、ドメイン名が別の Amazon CloudFront ディストリビューションの代替ドメインとして既に使用されている場合に発生します。

問題

カスタムドメインを作成すると、Amazon Cognito は AWS マネージド Amazon CloudFront ディストリビューションを作成します。ドメイン名は、1 つの Amazon CloudFront ディストリビューションでのみ使用できます。このエラーは、ドメイン名が別の Amazon CloudFront ディストリビューションの代替ドメインとして既に使用されている場合に発生します。

ソリューション

- オプション 1: Amazon Cognito カスタムドメインに別のドメイン名を使用します。
- オプション 2: Amazon Cognito でドメイン名を使用している場合は、別の Amazon CloudFront ディストリビューションで同じ名前を使用しません。

The specified SSL certificate doesn't exist

Amazon Cognito は Amazon CloudFront を使用します。Amazon CloudFront では、ユーザープールのリージョンに関係なく AWS リージョン、AWS Certificate Manager (ACM) 証明書が us-east-1 (バージニア北部) に存在する必要があります。

問題

Amazon Cognito は Amazon CloudFront を使用します。Amazon CloudFront では、ユーザープールのリージョンに関係なく AWS リージョン、AWS Certificate Manager (ACM) 証明書が us-east-1 (バージニア北部) に存在する必要があります。

ソリューション

- 証明書リージョンを確認する: ACM 証明書が us-east-1 リージョンにあることを確認します。
- 証明書の有効性を確認する: 選択した証明書の有効期限が切れていないことを確認します。
- インポートした証明書: 証明書を ACM にインポートした場合は、以下を確認します。
 - パブリック認証局によって発行されたものである。
 - 正しい証明書チェーンが含まれている。
- AWS KMS ポリシーチェック: ドメインを作成する IAM ユーザーまたはロールの AWS Key Management Service (AWS KMS) ポリシーの明示的な deny ステートメントにより、このエラーが発生する可能性があります。特に、kms:DescribeKey、kms:CreateGrant、または kms:* アクションの明示的な拒否を確認します。

Invalid refresh token エラー

問題

更新トークンを使用して、AdminInitiateAuth または InitiateAuth API オペレーションで Amazon Cognito ユーザープールから新しいアクセストークンや ID トークンを取得しようとする、Invalid refresh token エラーが発生します。

ソリューション

ユーザープールの設定と API の使用状況に応じて、次のトラブルシューティング手順を実装します。

- 一貫したアプリケーションクライアント ID を確認する: トークン更新のために AdminInitiateAuth または InitiateAuth API を呼び出すときは、最初の認証時に更新トークンの生成に使用したのとまったく同じアプリケーションクライアント ID を使用する必要があります。
- デバイスを確認する: ユーザープールで [デバイス追跡](#) を有効にしても、ユーザーのデバイスが未確認である場合は、まず [ConfirmDevice](#) API を呼び出す必要があります。ユーザーがデバイスを確認したら、更新トークンを交換できます。
- 更新リクエストにデバイスキーを含める: デバイス追跡が有効になっている場合は、REFRESH_TOKEN_AUTH フローの使用時に一意のデバイスキーを AuthParameter として含めます。

```
{
  "AuthFlow": "REFRESH_TOKEN_AUTH",
  "AuthParameters": {
    "REFRESH_TOKEN": "example_refresh_token",
    "SECRET_HASH": "example_secret_hash", // Required if your app client uses a
client secret
    "DEVICE_KEY": "example_device_key"
  }
}
```

- デバイス追跡に **USER_SRP_AUTH** を使用する: デバイス追跡を使用している場合、初期認証フローは USER_SRP_AUTH である必要があります。

詳細については、「[ユーザープール内のユーザーデバイスの使用](#)」を参照してください。

フェデレーションでの無効な SAML レスポンスエラー

ユーザーが SAML 2.0 を使用して Amazon Cognito にフェデレーションしようとする、さまざまな Invalid SAML response エラーや同様のエラーが発生します。これらのエラーは、属性マッピングの問題、証明書の問題、または設定の不一致が原因で発生する可能性があります。

Invalid user attributes: Required attribute

問題

ユーザープールで必須属性の値が欠落しているか、IdP がイミュータブルな属性を削除または更新しようとしています。

ソリューション

- ユーザープール設定に定義されている[必須属性](#)を確認します。
- ブラウザでネットワークキャプチャツールを使用して、SAML レスポンスを取得します。URL と base64 のデコードが必要になる場合があります。属性が SAML アサーションに存在することを確認します。
- ID プロバイダーにサインインし、プロバイダーから Amazon Cognito に送信している属性を確認します。IdP が正しい名前を使用して必須属性を送信するように設定されていることを確認します。
- 変更不可能な属性の場合は、次の AWS CLI コマンドを実行して識別します:

```
aws cognito-idp describe-user-pool --user-pool-id USER-POOL-ID --query 'UserPool.SchemaAttributes[?Mutable==`false`].Name'
```
- IdP の SAML 属性マッピングで、イミュータブルな Amazon Cognito 属性をターゲットとしているマッピングをすべて削除します。または、送信先属性を別の変更可能な属性に更新します。

Invalid SAML response received: SAML Response signature is invalid

問題

IdP が SAML 署名証明書を更新したため、SAML レスポンスの証明書と Amazon Cognito に保存されているメタデータファイルが一致しません。

ソリューション

1. IdP から最新のメタデータファイルをダウンロードします。
2. Amazon Cognito コンソールで、ユーザープールの [ソーシャルプロバイダーと外部プロバイダー] に移動して SAML プロバイダーを編集し、既存のメタデータファイルを新しくダウンロードしたファイルに置き換えます。

Audience restriction、または Application with identifier not found

問題

IdP で正しくないエンティティ ID が設定されているか、アサーションが別のユーザープールの Uniform Resource Name (URN) を使用しています。

ソリューション

1. コンソールの [概要] セクションから Amazon Cognito ユーザープール ID を取得します。
2. IdP のマネジメントコンソールで、ユーザープールの SAML アプリケーションのエンティティ ID を更新します。形式が `urn:amazon:cognito:sp:USER_POOL_ID` と一致するようにエンティティ ID を設定します。USER_POOL_ID を、前のステップのユーザープール ID に置き換えます。

An error was encountered with the requested page

問題

IdP に登録されているアサーションコンシューマーサービス (ACS) の URL が正しく設定されていないか、IdP が必要な POST バインディングを使用して SAML レスポンスを送信していません。

ソリューション

- IdP のマネジメントコンソールで、アプリケーションを更新して正しい ACS URL 形式を反映し、アプリケーションが HTTP POST バインディングを使用していることを確認します。
- デフォルトドメイン形式: `https://cognito-idp.Region.amazonaws.com/your user pool ID/saml2/idpresponse`
- カスタムドメイン形式: `https://auth.example.com/saml2/idpresponse`

Invalid relayState from identity provider

問題

RelayState パラメータが欠落しているか、無効であるか、IdP と Amazon Cognito の間で URL が一致していません。

ソリューション

- [サービスプロバイダーによって開始された \(SP によって開始された\)](#) フローの場合: 常にユーザープールの /oauth2/authorize エンドポイントで認証を開始します。ユーザーが初めて Amazon Cognito にアクセスしたときに RelayState パラメータを返さない SAML アサーションは、SP によって開始された有効なリクエストではありません。
- [ID プロバイダーによって開始された \(IdP によって開始された\)](#) フローの場合: IdP は、必要な形式 (redirect_uri=REDIRECT_URI&state=STATE) を使用して、/saml2/idpresponse エンドポイントへの SAML アサーションに RelayState パラメータを含める必要があります。

詳細については、「[ユーザープールによる SAML ID プロバイダーの使用](#)」を参照してください。

マネージドログインユーザーが MFA 要素を選択できない

問題

ユーザーは、マネージドログインでサインインするときに希望する MFA の方法を選択できないか、または希望する MFA の方法を入力するよう求められません。

ソリューション

Amazon Cognito は、特定のロジックに従い、ユーザープールの設定、ユーザー属性、アカウント復旧設定に基づいてユーザーが利用できる MFA 要素を決定します。例えば、E メールがプライマリのアカウント復旧方法として設定されている場合、ユーザーは E メール MFA を使用できません。

MFA 要素のデフォルトの選択を上書きするには、次のいずれかのアプローチを実装できます。

- パブリックアプリケーションの場合: 有効なアクセストークンで [SetUserMFAPreference](#) を使用して、ユーザーが希望する MFA を選択できるよう設定します。
- 管理者マネージドアプリケーションの場合: AWS 認証情報で [AdminSetUserMFAPreference](#) を使用してユーザー MFA 設定を構成する

どちらのオペレーションでも、ユーザーごとに SMS、E メール、TOTP MFA 方法を有効または無効にし、希望する方法として 1 つを設定できます。

詳細については、「[ユーザープールに MFA を追加します](#)」を参照してください。

パスワードなしのユーザーとパスキーユーザーは MFA を使用できません

問題

パスワードなしの認証方法またはパスキーでサインインするユーザーは、多要素認証を追加または使用できません。

ソリューション

これは意図された制限です。ユーザーが MFA 要素を使用するか、パスワードなしの要素でサインインするようユーザープールを設定できますが、両方を設定することはできません。MFA はパスワードベースの認証フローでのみ使用できます。

詳細については、「[ユーザープールによる認証](#)」を参照してください。

E メール/SMS 経由でパスワードリセットコードを受信できない

問題

ユーザーは、パスワードを忘れた場合のワークフローで、E メールまたは SMS を通じて検証コードを受信できません。

ソリューション

検証コードが受信できない理由はさまざまです。この問題のトラブルシューティングを行うには、次のチェックリストに従います。

- ユーザーの E メールのスパムフォルダと迷惑メールフォルダを確認します。
- ユーザープールにユーザーが存在することを確認します。
- ユーザーのステータスが `FORCE_CHANGE_PASSWORD` でないことを確認します。このステータスである場合、ユーザーは管理者によって作成されており、Amazon Cognito は、ユーザーが仮パスワードを使用してサインインするときにパスワードを設定するよう求めます。
- ユーザーの E メールまたは電話番号の属性が検証済みであることを確認します。
- SMS メッセージングのアカウント使用制限を確認する
- Amazon Simple Email Service (Amazon SES) のメッセージ送信クォータを確認します。
- SMS と Amazon SES の両方 (ユーザープールが [Amazon SES で E メールを送信] に設定されている場合) がサンドボックス外に移動されているかどうかを確認します。Amazon SES によって検証されていない、またはパスワードリセットコードを受信できないサンドボックス

ス、E メールアドレス、または電話番号から移動されなかった AWS End User Messaging SMS 場合。

- ユーザーがアクティブな MFA 要素を持っている場合は、同じ要素へのメッセージを生成しようとしていないことを確認します。例えば、E メール MFA がアクティブであるユーザーは、パスワードリセットコードを E メールで送信できません。

詳細については、「[Amazon Cognito ユーザープールの E メール設定](#)」および「[Amazon Cognito ユーザープール用の SMS メッセージ設定](#)」を参照してください。

パスワードのリセットは、未検証の復旧属性で失敗します。

Could not reset password for the account, please contact support or try again

問題

未検証の復旧方法または MFA 設定の競合が原因でパスワードをリセットしようとする、このエラーが発生します。エラーは、ユーザーの E メールまたは電話番号属性が未検証の場合、または MFA 設定が設定された復旧方法を使用できない場合に発生します。

ソリューション

ユーザープールのアカウント復旧設定と、影響を受けるユーザーの検証ステータスを確認します。

- 復旧設定の確認: ユーザープールで、サインイン > ユーザーアカウントの復旧に移動します。セルフサービスアカウントの復旧が有効になっていることを確認し、復旧メッセージ配信方法の設定を確認します。
- ユーザー属性の検証を確認する: 復旧方法の設定に一致する検証済みの E メールアドレスまたは電話番号がユーザーにあることを確認します。コンソールで、ユーザーのプロファイルに移動し、ユーザー属性 > 編集を選択し、適切な属性を検証済みとしてマークします。
- MFA の競合を解決する: E メールが優先される MFA メソッドを持つユーザーは、E メールでパスワードリセットコードを受信できず、SMS MFA を持つユーザーは SMS でコードを受信できません。Recovery メッセージ配信方法を更新して、利用可能な場合は SMS、利用可能な場合は E メールまたは E メール、それ以外の場合は SMS などの代替オプションを提供します。
- 管理検証: API オペレーション [AdminUpdateUserAttributes](#) を使用して、コンソールアクセスが利用できないときにプログラムでユーザー属性を検証します。

詳細については、「[パスワード、アカウント復旧、パスワードポリシー](#)」を参照してください。

SECRET_HASH エラー

問題

クライアントシークレットを持つアプリケーションクライアントへの認証 API リクエストが An error occurred (NotAuthorizedException) when calling the ForgotPassword operation: Client 1example23456789 is configured with secret but SECRET_HASH was not received. などのエラーを返す

ソリューション

アプリケーションは、現在のユーザー、アプリケーションクライアント、クライアントシークレットについて、SECRET_HASH を計算する必要があります。計算方法は次のとおりです。

```
Base64 ( HMAC_SHA256 ( "client secret", "Username" + "Client Id" ) )
```

クライアントシークレットを取得するには、以下の操作ができます。

1. Amazon Cognito コンソールを開き、[アプリケーションクライアント] メニューからアプリケーションクライアントに移動します。[アプリケーションクライアント情報] セクションで、[クライアントシークレット] を見つけます。[クライアントシークレットを表示] を選択すると、アプリケーションのクライアントシークレットが表示されます。
2. [DescribeUserPoolClient](#) リクエストを生成します。クライアントシークレットがレスポンスに含まれます。

詳細については、「[シークレットハッシュ 値の計算](#)」を参照してください。

Amazon Cognito コンソールが、新しいユーザープールにデフォルト設定を選択する

問題

コンソールで新しいユーザープールを設定すると、Amazon Cognito はいくつかのデフォルト設定を選択します。一部の設定は、ユーザープールの作成後に変更できません。情報に基づいた選択を行い、Amazon Cognito が自動的に何を選択したかを確認するにはどうすればよいですか？

ソリューション

Amazon Cognito コンソールの新しいユーザープールの設定は、迅速なテストとプロトタイプ作成のために設計されています。コンソールには、ユーザープールの作成後に変更できない、最も重要な設定オプションのみが表示されます。Amazon Cognito が自動的に設定する他のすべての設定は、後で変更できます。

次のようなアプローチを推奨します。

1. コンソールを使用して、テストユーザープールを作成し、実装を改善します。
2. 本番稼働用設定を決定したら、この設定をテストユーザープールに適用します。
3. [DescribeUserPool](#) および [DescribeUserPoolClient](#) API オペレーションを使用して、テスト済み設定の JSON テンプレートを生成します。
4. AWS SDKs、CDK、REST API、などのデプロイツールでこれらのテンプレート CloudFormation を使用して、本番稼働用リソースを作成します。

詳細については、「[ユーザープールの開始方法](#)」を参照してください。

その他のトラブルシューティングリソース

その他のトラブルシューティングガイダンスとコミュニティから提供されたソリューションについては、以下の外部リソースも参照してください。

- [AWS re:Post Amazon Cognito コミュニティ](#) - コミュニティの質問とソリューションを参照する
- [AWS ナレッジセンターの Amazon Cognito 記事](#) - 厳選されたトラブルシューティング記事

Amazon Cognito リソースのタグ付け

タグは、リソース AWS に割り当てるメタデータラベルです AWS。各タグは、キーと値から構成されます。ユーザーが割り当てるタグでは、ユーザーがキーと値を定義します。たとえば、1 つのリソースのキーを `stage` と定義し、値を `test` と定義します。

タグは、以下のことに役立ちます。

- AWS リソースを特定して整理します。多くの AWS サービスはタグ付けをサポートしているため、異なるサービスのリソースに同じタグを割り当てることができます。タグは、どのリソースが関連しているかを示すのに役立ちます。例えば、Amazon DynamoDB テーブルに割り当てるタグと同じものを Amazon Cognito ユーザープールに割り当てることができます。
- AWS コストを追跡します。AWS Billing and Cost Management ダッシュボードでこれらのタグをアクティブ化できます。AWS はコスト配分タグを使用してコストを分類し、毎月のコスト配分レポートを配信します。詳細については、「AWS Billing ユーザーガイド」の「[Use cost allocation tags](#)」(コスト配分タグの使用) を参照してください。
- リソースに割り当てられたタグに基づいて、リソースへのアクセスをコントロールします。AWS Identity and Access Management (IAM) ポリシーの条件でキーと値を指定することによってアクセスをコントロールできます。例えば、ユーザープールの更新をユーザーに許可しても、そのユーザープールに `owner` タグと、そのユーザーの名前の値がある場合のみに限定することができます。詳細については、IAM ユーザーガイドの「[タグを使用したアクセス制御](#)」を参照してください。

AWS Command Line Interface または Amazon Cognito API を使用して、ユーザープールと ID プールの両方のタグを追加、編集、または削除できます。Amazon Cognito コンソールを使用して、ユーザープールのタグを管理することもできます。

タグの使用法のヒントについては、AWS の回答ブログの[AWS タグ付け戦略](#)記事を参照してください。

以下のセクションでは、Amazon Cognito のタグに関する詳細が説明されています。

Amazon Cognito でサポートされるリソース

Amazon Cognito の以下のリソースがタグ付けをサポートしています。

- ユーザープール

- ID プール

タグの制限

Amazon Cognito リソースのタグには、以下の制限が適用されます。

- リソースに割り当てることができるタグの最大数: 50
- キーの最大長 – 128 文字 (Unicode)
- 値の最大長 – 256 文字 (Unicode)
- キーと値の有効な文字 – a~z、A~Z、0~9、スペース、および特殊文字 (_ . : / = + - @)
- キーと値では大文字と小文字が区別されます
- aws: をキーのプレフィックスとして使用しないでください。AWS 用に予約済みです。

Amazon Cognito コンソールを使用したタグの管理

Amazon Cognito コンソールを使用して、ユーザープールに割り当てられたタグを管理することができます。

ユーザープールにタグを追加するには

1. [\[Amazon Cognito console\]](#) (Amazon Connect コンソール) に移動します。プロンプトが表示されたら、AWS 認証情報を入力します。
2. [User Pools] (ユーザープール) を選択します。
3. リストから既存のユーザープールを選択するか、[ユーザープールを作成](#)します。
4. [設定] メニューを選択し、[タグ] タブを見つけます。
5. [Add tags] (タグを追加する) を選択し、最初のタグを追加します。以前このユーザープールにタグを割り当てたことがある場合は、[Manage tags] (タグの管理) を選び、[Add another] (別のタグを追加) を選択します。
6. [Tag Key] (タグキー) と [Tag Value] (タグ値) に値を入力します。
7. 追加するタグごとに、[Add another tag] (別のタグを追加) を選択します。
8. タグの追加を完了したら、[Save changes] (変更の保存) を選択します。

アイデンティティプールにタグを付けるには、[ID プール] メニューに移動し、アイデンティティプールを選択または作成します。[ID プールのプロパティ] タブで、[タグ] を見つけます。[タグを追加] を選択します。

AWS CLI 例

AWS CLI には、Amazon Cognito ユーザープールと ID プールに割り当てるタグの管理に役立つコマンドが用意されています。

タグの割り当て

次のコマンドを使用して、既存のユーザープールと ID プールにタグを割り当てます。

Example **tag-resource** ユーザープール向けのコマンド

コマンドの `cognito-idp` セット内の [tag-resource](#) 使用して、ユーザープールにタグを割り当てます。

```
$ aws cognito-idp tag-resource \  
> --resource-arn user-pool-arn \  
> --tags Stage=Test
```

このコマンドには次のパラメータが含まれます。

- `resource-arn` – タグを割り当てるユーザープールの Amazon リソースネーム (ARN)。ARN を調べるには、Amazon Cognito コンソールでユーザープールを選択し、[General settings] (一般設定) タブの [Pool ARN] (プール ARN) 値を表示します。
- `tags` — 形式 *key=value* のタグのキーバリューペアです。

一度に複数のタグを割り当てるには、カンマ区切りリストで指定します。

```
$ aws cognito-idp tag-resource \  
> --resource-arn user-pool-arn \  
> --tags Stage=Test, CostCenter=80432, Owner=SysEng
```

Example **tag-resource** ID プール向けのコマンド

コマンドの `cognito-identity` セット内の [tag-resource](#) 使用して、ID プールにタグを割り当てます。

```
$ aws cognito-identity tag-resource \  
> --resource-arn identity-pool-arn \  
> --tags Stage=Test
```

このコマンドには次のパラメータが含まれます。

- **resource-arn** – タグを割り当てる ID プールの Amazon リソースネーム (ARN)。ARN を調べるには、Amazon Cognito コンソールで ID プールを選択し、[Edit identity pool] (ID プールの編集) をクリックします。その後、[Identity pool ID] (ID プールの ID) で [Show ARN] (ARN の表示) をクリックします。
- **tags** — 形式 *key=value* のタグのキーバリューペアです。

一度に複数のタグを割り当てるには、カンマ区切りリストで指定します。

```
$ aws cognito-identity tag-resource \  
> --resource-arn identity-pool-arn \  
> --tags Stage=Test, CostCenter=80432, Owner=SysEng
```

タグの表示

次のコマンドを使用して、ユーザープールと ID プールに割り当てたタグを表示します。

Example **list-tags-for-resource** ユーザープール向けのコマンド

コマンドの `cognito-idp` セット内の [list-tags-for-resource](#) 使用して、ユーザープールにタグを割り当てます。

```
$ aws cognito-idp list-tags-for-resource --resource-arn user-pool-arn
```

Example **list-tags-for-resource** ID プール向けのコマンド

コマンドの `cognito-identity` セット内の [list-tags-for-resource](#) 使用して、ID プールにタグを割り当てます。

```
$ aws cognito-identity list-tags-for-resource --resource-arn identity-pool-arn
```

タグの削除

以下のコマンドを使用して、ユーザープールと ID プールからタグを削除します。

Example **untag-resource** ユーザープール向けのコマンド

コマンドの `cognito-idp` セット内の [untag-resource](#) を使用して、ユーザープールにタグを割り当てます。

```
$ aws cognito-idp untag-resource \  
> --resource-arn user-pool-arn \  
> --tag-keys Stage CostCenter Owner
```

`--tag-keys` パラメータには、1 つ以上のタグキーを指定します。タグ値は含みません。キーをスペースで区切ります。

Example **untag-resource** ID プール向けのコマンド

コマンドの `cognito-identity` セット内の [untag-resource](#) を使用して、ID プールにタグを割り当てます。

```
$ aws cognito-identity untag-resource \  
> --resource-arn identity-pool-arn \  
> --tag-keys Stage CostCenter Owner
```

`--tag-keys` パラメータには、1 つ以上のタグキーを指定します。タグ値は含みません。

Important

ユーザーまたは ID プールの削除後、削除されたプールに関連するタグは、削除から最大 30 日までコンソールまたは API コールに表示できます。

リソース作成時のタグの適用

次のコマンドを使用して、ユーザープールまたは ID プールを作成するときにタグを割り当てます。

Example **create-user-pool** コマンドとタグ

[create-user-pool](#) コマンドを使用してユーザープールを作成するときは、`--user-pool-tags` パラメータと共にタグを指定できます。

```
$ aws cognito-idp create-user-pool \  
> --pool-name user-pool-name \  
> --user-pool-tags Stage CostCenter Owner
```

```
> --user-pool-tags Stage=Test, CostCenter=80432, Owner=SysEng
```

タグのキーバリューペアは、*key=value*形式が必須です。複数のタグを追加する場合は、カンマで区切りを付けたリストで指定します。

Example **create-identity-pool** コマンドとタグ

[create-identity-pool](#) コマンドを使用して ID プールを作成するときは、`--identity-pool-tags` パラメータと共にタグを指定できます。

```
$ aws cognito-identity create-identity-pool \  
> --identity-pool-name identity-pool-name \  
> --allow-unauthenticated-identities \  
> --identity-pool-tags Stage=Test, CostCenter=80432, Owner=SysEng
```

タグのキーバリューペアは、*key=value*形式が必須です。複数のタグを追加する場合は、カンマで区切りを付けたリストで指定します。

Amazon Cognito API を使用したタグの管理

以下のアクションを Amazon Cognito API で使用することで、ユーザープールと ID プールのタグを管理できます。

ユーザープールタグの API アクション

次の API アクションを使用して、ユーザープールのタグを割り当て、表示、削除します。

- [TagResource](#)
- [ListTagsForResource](#)
- [UntagResource](#)
- [CreateUserPool](#)

ID プールタグの API アクション

次の API アクションを使用して、ID プールのタグを割り当て、表示、削除します。

- [TagResource](#)
- [ListTagsForResource](#)

- [UntagResource](#)
- [CreateIdentityPool](#)

Amazon Cognito のクォータ

Amazon Cognito には、アカウントで実行できる操作の最大数のデフォルトクォータ (以前は制限と呼ばれていました) があります。Amazon Cognito では、Amazon Cognito リソースの最大数と最大サイズのクォータがあります。

各 Amazon Cognito クォータは、1 つの AWS リージョン 内のリクエストの最大量を表します。AWS アカウント。例えば、アプリは、米国東部 (バージニア北部) のすべてのユーザープールに対して、UserAuthentication オペレーションのデフォルトクォータ (RPS) レートまでの API リクエストを行うことができます。アジアパシフィック (東京) のアプリは、自分のリージョンのすべてのユーザープールに対して同じ量のリクエストを生成できます。AWS は、一度に 1 つのリージョンでのみクォータ引き上げリクエストを付与できます。米国東部 (バージニア北部) でクォータを引き上げても、アジアパシフィック (東京) の最大リクエストレートには影響しません。

トピック

- [API リクエストレートクォータについて](#)
- [API リクエストレートクォータ](#)
- [Amazon Cognito ユーザープール API オペレーションカテゴリとリクエストレートクォータ](#)
- [Amazon Cognito ID プール \(フェデレーティッドアイデンティティ\) API オペレーションリクエストレートのクォータ](#)
- [リソースの番号とサイズのクォータ](#)

API リクエストレートクォータについて

クォータのカテゴリ

Amazon Cognito は API オペレーションの最大リクエストレート適用します。Amazon Cognito で利用できる API オペレーションの詳細については、[ユーザープール](#)と[アイデンティティプール](#)に関する API リファレンスガイドを参照してください。ユーザープールについては、これらの操作は、UserAuthentication または UserCreation などの一般的なユースケースのカテゴリに分類されます。ユーザープール API オペレーションのカテゴリ別リストについては、「[Amazon Cognito ユーザープール API オペレーションカテゴリとリクエストレートクォータ](#)」を参照してください。

[Service Quotas コンソール](#)では、カテゴリユーザープール別、アイデンティティプール別にクォータ使用状況を追跡できます。Amazon Cognito ユーザープールのリクエストレートがクォータを超え

ている場合は、追加の容量を購入できます。[Service Quotas コンソール](#)で、ユーザープールのカテゴリ別クォータの使用状況と購入クォータの増加を追跡できます。

操作のクォータは、カテゴリ内のすべての操作について、1 秒あたりのリクエスト (RPS) の最大数として定義されます。Amazon Cognito ユーザープールサービスは、各カテゴリのすべての操作にクォータを適用します。例えば、カテゴリ `UserCreation` には `SignUp`、`ConfirmSignUp`、`AdminCreateUser`、および `AdminConfirmSignUp` の 4 つのオペレーションが含まれます。これには合計で 50 RPS のクォータが割り当てられています。同時に複数のオペレーションが実行される場合は、オペレーションごとに最大 50 の RPS を別々に、または合わせて呼び出すことができます。

Note

カテゴリクォータはユーザープールにのみ適用されます。Amazon Cognito は、各 ID プールクォータを 1 つのオペレーションに適用します。カテゴリごとのリクエストレートクォータとオペレーションごとのリクエストレートクォータの両方で、は、1 つのリージョン AWS アカウントの内のすべてのユーザープールまたは ID プールからのすべてのリクエストの集計レート AWS を測定します。

リクエストレートの特別な取り扱いを使用した Amazon Cognito ユーザープール API オペレーション

オペレーションクォータは、カテゴリレベルでのリクエストの合計数で測定され、適用されますが、特別な取扱いルールが適用される `AdminRespondToAuthChallenge` および `RespondToAuthChallenge` オペレーションに対しては例外になります。

`UserAuthentication` カテゴリには、Amazon Cognito ユーザープール API のうち、`AdminInitiateAuth`、`InitiateAuth`、`AdminRespondToAuthChallenge`、`RespondToAuthChallenge` の 4 つのオペレーションが含まれます。さらに、ホストされた UI のユーザー認証はこのクォータにカウントされます。`InitiateAuth` および `AdminInitiateAuth` オペレーションは、カテゴリクォータごとに測定され、施行されます。マッチングオペレーション `RespondToAuthChallenge` および `AdminRespondToAuthChallenge` は、別のクォータの対象であり、`UserAuthentication` カテゴリの制限の 3 倍です。クォータが拡大されているので、アプリケーションで設定される複数の認証チャレンジに対応できます。このクォータは、ほとんどのユースケースに十分対応できます。アプリケーションが認証チャレンジに最大 3 つの応答を行った後は、追加のリクエストは `UserAuthentication` カテゴリクォータにカウントされます。[多要素](#)

[認証 \(MFA\)](#)、[デバイス認証](#)、[カスタム認証](#)はすべて、ユーザープールに設計する可能性のあるチャレンジプロンプトの例です。

例えば、UserAuthentication カテゴリのクォータが 80 RPS である場合、RespondToAuthChallenge または AdminRespondToAuthChallenge を最大 240 RPS (3 x 80 RPS) を呼び出すことができます。ユーザープールが認証ごとに 4 ラウンドのチャレンジを要求し、1 秒あたり 70 人のユーザーがサインインした場合、合計 RespondToAuthChallenge は 280 RPS (70 x 4) で、クォータを 40 RPS 上回ります。40 RPS が 70 回の InitiateAuth コールに追加されたことで、UserAuthentication カテゴリの総使用量は 110 RPS (40 + 70) になります。この値は、80 RPS に設定されたカテゴリクォータを 30 RPS 超えるため、Amazon Cognito はアプリケーションからのリクエストをスロットリングします。

月次のアクティブユーザー

Amazon Cognito がユーザープールの請求を計算すると、月間アクティブユーザー (MAU) ごとに料金が請求されます。クォータ引き上げリクエストの計画では、MAU 数の現在値と予測値を考慮してください。暦月内に、ユーザーに関連した ID オペレーションがある場合、そのユーザーは MAU としてカウントされます。SAML または OIDC フェデレーションを使用して、[フェデレーションユーザーをローカルユーザーにリンク](#)すると、ローカルユーザーは、ユーザーが直接サインインしたか、フェデレーション経由でサインインしたかに関係なく、エンタープライズディレクトリ MAU (EnterpriseMAU) としてカウントされます。詳細については、「[Amazon Cognito の料金](#)」を参照してください。

- ユーザーのサインアップまたは管理者による作成。[ユーザーの CSV インポート](#)は、MAU 数に影響しません。
- ユーザーアカウントの確認または属性の検証。
- サインインとチャレンジレスポンス。現在サインインしているユーザーのアクセストークンを使用して認可したオペレーションは、MAU 数に加算されません。ただし、サインインによってアクセストークンが生成されるため、これらのオペレーションは関連するユーザーが MAU であることを示します。
- サインアウトとトークンの取り消し。
- 管理者としてのユーザーパスワードのセルフサービスリセットおよび設定。管理者としてユーザーパスワードをリセット ([AdminResetUserPassword](#)) しても、MAU 数には加算されません。
- ユーザー属性またはグループメンバーシップの変更。
- ユーザーの詳細な属性を管理者としてクエリ。

 Note

ユーザーの詳細な属性を管理者としてクエリというカテゴリには、API オペレーション [AdminGetUser](#) が含まれますが、[ListUsers](#) は含まれません。大規模なuser-by-userクエリは、AWS 請求に大きな影響を与える可能性があります。追加コストを回避するには、ListUsers を使用してユーザーデータを収集するか、外部データベースにユーザー情報を保存します。

アクティブなユーザーによる追加のセッション、または暦月内にアクティブでなかったユーザーに対しては課金されません。ユーザープールの機能プランを、月内に利用可能なライト、エッセンシャル、プラスのオプション間で変更した場合、請求月額は、階層ごとの月間アクティブユーザー数 (MAU) の合計から計算されます。各 MAU は、ユーザーがアクティブになった時点での最高額の階層に割り当てられます。例えば、次のようになります。

1. 月の初めに、ユーザープールはプラス機能プランになっています。
2. ユーザー A は月の初日にサインインします。
3. ユーザー B は、月の初日と最終日にサインインします。
4. 月の 10 日に、機能プランをエッセンシャルに切り替えます。
5. ユーザー C は、月の最終日にサインインします。

このシナリオの場合、ユーザー A とユーザー B はプラス MAU で、ユーザー C はエッセンシャル MAU になります。

ライト MAU

ユーザープールがライト機能プランであった月に少なくとも 1 回アクティブになり、ユーザープールがエッセンシャルプランまたはプラスプランであったときに一度もアクティブにならなかったユーザー。

エッセンシャル MAU

ユーザープールがエッセンシャル機能プランであった月に少なくとも 1 回アクティブになり、ユーザープールがプラスプランであったときに一度もアクティブにならなかったユーザー。

プラス MAU

ユーザープールがプラスプランであった月に少なくとも 1 回アクティブになったユーザー。

詳細については、「[ユーザープールの機能プラン](#)」を参照してください。

API リクエストレートクォータ

クォータ要件を特定する

Important

UserAuthentication、UserCreation、または AccountRecovery のようなカテゴリに対して Amazon Cognito クォータを引き上げた場合に、他の AWS のサービスのクォータを引き上げる必要が生じることがあります。例えば、Amazon Cognito が Amazon Simple Notification Service (Amazon SNS) または Amazon Simple Email Service (Amazon SES) で送信するメッセージは、これらのサービスでリクエストレートのクォータが不十分な場合に失敗することがあります。

クォータ要件を計算するには、特定の期間内にアプリケーションとやり取りするアクティブなユーザーの数を判断します。例えば、アプリケーションに 8 時間の期間内に平均 100 万人のアクティブユーザーによるサインインがあることを想定する場合、1 秒あたり平均 35 人のユーザーを認証する必要があります。

さらに、平均的なユーザーセッションが 2 時間であると仮定し、トークンが 1 時間後に期限切れになるように設定している場合、各ユーザーは、セッション中にトークンを 1 回更新する必要があります。この負荷をサポートするために UserAuthentication カテゴリに必要な平均クォータは 70 RPS になります。

8 時間の期間内でのユーザーのサインイン頻度の変動を考慮して、ピーク対平均の比を 3:1 と仮定する場合、望ましい UserAuthentication クォータは 200 RPS にする必要があります。

Note

ユーザーアクションごとに複数のオペレーションを呼び出す場合は、カテゴリレベルで個々のオペレーションのコールレートを合計する必要があります。

クォータ制限のリクエストレートを最適化する

API レート制限を増やすと AWS 請求にコストがかかるため、クォータの引き上げをリクエストする前に使用量モデルの調整を検討してください。以下は、リクエストレートを最適化するアプリケーションアーキテクチャの例です。

バックオフ待機期間後に再試行する

API コールごとにエラーをキャッチし、バックオフ期間後に再試行することができます。バックオフアルゴリズムは、ビジネスのニーズと負荷に応じて調整できます。Amazon SDK には、組み込みの再試行ロジックがあります。詳細については、[「構築するツール」を参照してください AWS。](#)

頻繁に更新される属性に外部データベースを使用する

アプリケーションが、カスタム属性の読み取りと書き込みにユーザープールへの複数のコールを必要とする場合は、外部ストレージを使用します。優先データベースを使用してカスタム属性を保存する、またはキャッシュレイヤーを使用してサインイン時にユーザープロフィールをロードすることができます。このプロフィールは、ユーザープールからユーザープロフィールを再ロードしなくても、必要に応じてキャッシュから参照することができます。

クライアント側で JSON ウェブトークン (JWT) を検証する

アプリケーションは、JWT トークンを信頼する前にそれらを検証する必要があります。トークンの署名と有効期間は、API リクエストをユーザープールに送信することなく、クライアント側で検証できます。トークンが検証されたら、トークン内のクレームを信頼して、さらに多くの getUser API コールを実行せずにクレームを使用することができます。詳細については、[「JSON Web トークンの検証」](#)を参照してください。

待合室でウェブアプリケーションへのトラフィックを調整する

試験の受験やライブイベントへの参加など、時間が限られているイベント中に多数のユーザーがサインインするトラフィックが予想される場合は、セルフスロットリングメカニズムを使用してリクエストトラフィックを最適化できます。例えば、セッションが利用可能になるまでユーザーが待機できる待合室を設定して、使用可能な容量がある間にリクエストを処理できます。待合室のリファレンスアーキテクチャについては、[「AWS Virtual Waiting Room」](#)を参照してください。

JWT のキャッシング

アクセストークンの有効期限が切れるまで再利用します。API Gateway でトークンキャッシングを使用するフレームワークの例については、[「ユーザープールトークンの有効期限とキャッ](#)

[シユの管理](#)」を参照してください。ユーザー情報をクエリする API リクエストを生成する代わりに、ID トークンを有効期限が切れるまでキャッシュし、キャッシュからユーザー属性を読み取ります。

での API リクエストレートの操作の詳細については AWS、「[ワークロードでの API スロットリングの管理とモニタリング](#)」を参照してください。AWS 請求書にコストを追加する Amazon Cognito オペレーションの最適化については、「」を参照してください[のコスト管理](#)。

クォータの使用状況を追跡する

Amazon Cognito では、アカウントレベルでの各 API オペレーションカテゴリの Amazon CloudWatch の CallCount および ThrottleCount メトリクスが生成されます。お客様が実行するカテゴリ関連のコールの合計数を追跡するには、CallCount を使用できます。カテゴリ関連のスロットルされたコールの合計数は、ThrottleCount を使用して追跡できます。カテゴリ内のコールの合計数を計上するには、Sum 統計で CallCount および ThrottleCount メトリクスを使用できます。詳細については、「[Amazon CloudWatch メトリクスを使用する](#)」を参照してください。

サービスクォータをモニタリングする場合、使用率は使用中のサービスクォータの割合です。例えば、クォータの値が 200 リソースで 150 リソースが使用中の場合、使用率は 75% です。使用状況は、サービスクォータに対する使用中のリソースまたはオペレーションの数です。

CloudWatch メトリクスを使用した使用状況の追跡

CloudWatch を使用して、Amazon Cognito ユーザープールの使用率メトリクスを追跡し、収集することができます。CloudWatch ダッシュボードには AWS のサービス、使用するすべてのメトリクスが表示されます。CloudWatch を使用すると、メトリクスアラームを作成して通知を受けたり、モニタリングしている特定のリソースを変更したりできます。CloudWatch メトリクスの詳細については、「[CloudWatch と Service Quotas でのクォータと使用状況の追跡](#)」を参照してください。

Service Quotas メトリクスを使用した使用率の追跡

Amazon Cognito ユーザープールは、サービスクォータの使用状況を表示して管理するためのコンソールインターフェイスである Service Quotas と統合されています。Service Quotas コンソールでは、特定のクォータの値を調べることで、モニタリング情報を表示すること、クォータの引き上げをリクエストすること、および CloudWatch アラームをセットアップすることができます。アカウントがアクティブ状態になってからしばらくすると、リソース使用率のグラフを表示できます。

[Amazon Cognito ユーザープール](#)用と [Amazon Cognito アイデンティティプール](#)用の Service Quotas コンソールの [適用されたアカウントレベルのクォータ値] 列には、現在のクォータが表示されま

す。[使用率] 列には、現在のクォータ使用率が表示されます。調整可能な Amazon Cognito ユーザープールの 1 秒あたりのリクエスト (RPS) クォータには、現在の使用状況が表示されます。また、Service Quotas コンソールは、CloudWatch メトリクスに移動して、選択したクォータメトリクスを詳しく調べることもできます。Service Quotas コンソールでのクォータの表示に関する詳細については、「[Viewing service quotas](#)」を参照してください。

月間アクティブユーザー数 (MAU) を追跡する

ユーザープール内の月間アクティブユーザー (MAU) の数は、リクエストレートクォータの引き上げに関する計画に対して重要なデータを提供します。API リクエストレートを、特定の期間においてアクティブであったユーザーの数と比較できます。このデータがあれば、アプリケーションのアクティブユーザーの増加が、使用状況モデルのクォータにどのように影響するかを計算できます。例えば、米国西部 (オレゴン) 内のアプリケーション全体で 1 か月のアクティブユーザーが 200 万人になり、UserAuthentication カテゴリが 120 リクエスト/秒 (RPS) のデフォルトクォータで時折スロットリングエラーを受け取ったとします。広告キャンペーンが成功する前の 1 か月間は、100 万 MAU であって、アプリケーションが 80 RPS を超えることはなかったとします。新しいテレビスポットを実施した結果として同様の急増が予想される場合は、ユーザー増加分の 100 万人に対応するために追加の 40 RPS を購入して、調整後のクォータが 160 RPS になるようにできます。

MAU を確認するには

[\[AWS Billing コンソール\]](#) にアクセスし、直近の請求書を確認します。[サービス別料金] において、[Cognito] でフィルタリングすると、その請求期間の MAU の内訳を表示できます。

クォータ引き上げのリクエスト

Amazon Cognito には、それぞれのユーザープールと ID プールで実行できる 1 秒あたりのオペレーションの最大数に対するクォータがあります AWS リージョン。調整可能な Amazon Cognito ユーザープール API リクエストレートクォータの引き上げを購入できます。現在のクォータを確認し、Service Quotas コンソールで、または Service Quotas API オペレーションの `ListAWSDefaultServiceQuotas` と `RequestServiceQuotaIncrease` を用いて引き上げを購入します。

- Service Quotas コンソールを使用してクォータの引き上げを購入するには、「Service Quotas ユーザーガイド」の「[Requesting a API quota increase](#)」を参照してください。
- AWS は、クォータ引き上げリクエストを 10 日以内に完了することを目標としています。ただし、いくつかの考慮事項により、リクエストの処理期間が 10 日を超える場合があります。例えば、リクエストによっては Amazon Cognito が追加のハードウェア容量をプロビジョニングする必要が生じたり、リクエストボリュームの季節的な増加により遅延が発生する場合があります。

- Service Quotas でクォータが利用できない場合は、[Service limit increase フォーム](#)を使用してください。

Important

クォータは、調整可能なもののみを引き上げることができます。クォータ容量の引き上げ分を購入する必要があります。クォータ引き上げ分の料金については、「[Amazon Cognito 料金](#)」を参照してください。

Amazon Cognito ユーザープール API オペレーションカテゴリとリクエストレートクォータ

Amazon Cognito には、[認可モデルが異なる](#) API オペレーションのクラスが重複しているため、各オペレーションはいずれかのカテゴリに属します。各カテゴリには、アカウント内の 1 つの AWS リージョン 内のすべてのユーザープールで、すべてのメンバー API オペレーションに対して独自のプールされたクォータがあります。調整可能なカテゴリのクォータについてのみ、引き上げをリクエストできます。詳細については、「[クォータ引き上げのリクエスト](#)」を参照してください。クォータ調整は、単一リージョンのアカウント内のユーザープールに適用されます。Amazon Cognito では、一部のカテゴリ³でのオペレーションを制限し、ユーザープールごとのリクエスト数/秒 (RPS) を 5 にしています。デフォルトクォータ (RPS) は、AWS アカウント内のすべてのユーザープールに追加で適用されます。

Note

各カテゴリのクォータは、月間アクティブユーザー数 (MAU) で測定されます。MAU が 200 万未満の AWS アカウント は、デフォルトのクォータ内で動作できます。MAU が 100 万未満で、Amazon Cognito がリクエストをスロットリングしている場合は、アプリケーションの最適化を検討してください。詳細については、「[クォータ制限のリクエストレートを最適化する](#)」を参照してください。

カテゴリオペレーションクォータは、1 つの AWS リージョン内のユーザープールのすべてのユーザーに適用されます。Amazon Cognito は、アプリが 1 人のユーザーに対して生成できるリクエストの数にも制限を設けています。ユーザーごとの API リクエストを以下のテーブルに示すとおり、制限する必要があります。

Amazon Cognito ユーザープールのユーザーあたりのリクエストレートクォータ

運用	ユーザーあたりのオペレーション数 (1 秒あたりのオペレーション数)
読み込みユーザープロフィール 例: GetUser、GetDevice 、 InitiateAuth 、 RespondToAuthChallenge	10
書き込みユーザープロフィール 例: UpdateUserAttributes 、 SetUserSettings	10

次の表に示すとおり、カテゴリ別の API リクエストを制限する必要があります。

Amazon Cognito ユーザープールのカテゴリごとのリクエストレートクォータ

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserAuthentication	ユーザーを認証 (サインイン) するオペレーション。	120	はい
• InitiateAuth • InitiateAuth または トークンエンドポイント によるトークンの更新 • RespondToAuthChallenge¹ • AdminInitiateAuth • AdminRespondToAuthChallenge¹	これらのオペレーションは リクエストレートの特別な取り扱いを使用した Amazon Cognito ユーザープール API オペレーション の対象となります。		

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
認可コードまたは暗黙的な付与²におけるローカルユーザーのマネージドログインまたはクラシックのホストされた UI サインインと MFA			
UserCreation SignUp ConfirmSignUp AdminCreateUser AdminConfirmSignUp	Amazon Cognito ローカルユーザーを作成または確認するオペレーション。これは、Amazon Cognito ユーザープールによって直接作成され、検証されるユーザーです。	50	はい
UserFederation サードパーティー ID プロバイダーを使用してユーザーを Amazon Cognito ユーザープールにフェデレート (認証) するオペレーション。	ユーザープールフェデレーションエンドポイントに IdP レスポンスを送信するオペレーション。IdP トークンを生成する OIDC またはソーシャルプロバイダーのオペレーション、およびすべての SAML リクエストが、このクォータに寄与します。	25	はい

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserAccountRecovery • ChangePassword • ConfirmForgotPassword • ForgotPassword • AdminResetUserPassword • AdminSetUserPassword • RespondToAuthChallenge¹ • AdminRespondToAuthChallenge¹ • マネージドログインのパスワードのリセット	ユーザーのアカウントを回復する、またはユーザーのパスワードを変更もしくは更新するオペレーション。	30	いいえ
UserRead • AdminGetUser • GetUser	ユーザープールからユーザーを取得する操作。	120	はい

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserUpdate • AdminAddUserToGroup • AdminDeleteUserAttributes • AdminUpdateUserAttributes • AdminDeleteUser • AdminDisableUser • AdminEnableUser • AdminLinkProviderForUser • AdminDisableProviderForUser • VerifyUserAttribute • DeleteUser • DeleteUserAttributes • UpdateUserAttributes • AdminUserGlobalSignOut • GlobalSignOut • AdminRemoveUserFromGroup	ユーザーとユーザー属性を管理するために使用するオペレーション。	25	いいえ
UserToken • RevokeToken	トークン管理のオペレーション	120	はい

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserResourceRead AdminGetDevice AdminListGroupsWithUser AdminListDevices GetDevice ListDevices GetUserAttributeVerificationCode ResendConfirmationCode AdminListUserAuthEvents	記憶されているデバイスやグループメンバーシップなどのユーザーリソース情報を Amazon Cognito から取得するオペレーション。	50	はい

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserResourceUpdate • AdminForgetDevice • AdminUpdateAuthEventFeedback • AdminSetUserMFAPreference • AdminSetUserSettings • AdminUpdateDeviceStatus • UpdateDeviceStatus • UpdateAuthEventFeedback • ConfirmDevice • SetUserMFAPreference • SetUserSettings • VerifySoftwareToken • AssociateSoftwareToken • ForgetDevice	記憶されているデバイスやグループメンバーシップなど、ユーザーのリソース情報を更新するオペレーション。	25	いいえ
UserList • ListUsers • ListUsersInGroup	ユーザーのリストを返す操作。	30	いいえ

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserPoolRead • DescribeUserPool • ListUserPools	ユーザープールを読み取る操作。	15	いいえ
UserPoolUpdate • CreateUserPool • UpdateUserPool • DeleteUserPool	ユーザープールを作成、更新、または削除するオペレーション。	15	いいえ

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserPoolResourceRead • DescribeIdentityProvider • DescribeResourceServer • DescribeUserImportJob • DescribeUserPoolDomain • GetCSVHeader • GetGroup • GetSigningCertificate • GetIdentityProviderByIdentifier • GetUserPoolMfaConfig • ListGroups • ListIdentityProviders • ListResourceServers • ListTagsForResource • ListUserImportJobs • DescribeRiskConfiguration • GetUICustomization	ユーザープールからグループやリソースサーバーなどのリソースに関する情報を取得するオペレーション。 ³	20	いいえ

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
DescribeManagedLogInBranding DescribeManagedLogInBrandingByClient			

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
UserPoolResourceUpdate • AddCustomAttribute • CreateGroup • CreateIdentityProvider • CreateResourceServer • CreateUserImportJob • CreateUserPoolDomain • DeleteGroup • DeleteIdentityProvider • DeleteResourceServer • DeleteUserPoolDomain • SetUserPoolMfaConfig • StartUserImportJob • StopUserImportJob • UpdateGroup • UpdateIdentityProvider • UpdateResourceServer	ユーザープールのグループやリソースサーバーなどのリソースを変更するオペレーション。 ³	15	いいえ

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
• UpdateUserPoolDomain • SetRiskConfiguration • SetUICustomization • TagResource • UntagResource • CreateManagedLoginBranding • UpdateManagedLoginBranding • DeleteManagedLoginBranding			
UserPoolClientRead • DescribeUserPoolClient • ListUserPoolClients	ユーザープールクライアントに関する情報を取得するオペレーション。 ³	15	いいえ
UserPoolClientUpdate • CreateUserPoolClient • DeleteUserPoolClient • UpdateUserPoolClient	ユーザープールクライアントを作成、更新、削除するオペレーション。 ³	15	いいえ

Category	説明	デフォルトクォータ (RPS)	引き上げ可能
ClientAuthentication トークンエンドポイントに対する client_credentials 付与タイプのリクエスト	マシン間リクエストの承認に使用される認証情報を生成するオペレーション	150	いいえ

¹ A RespondToAuthChallenge または AdminRespondToAuthChallenge レスポンスで、NEW_PASSWORD_REQUIRED の ChallengeName があるものは、UserAccountRecovery カテゴリにカウントされます。他のすべてのチャレンジレスポンスは、UserAuthentication カテゴリにカウントされます。

² サインイン中のマネージドログインまたはクラシックのホストされた UI のオペレーションごとに 1 つのリクエストとして、クォータにカウントされます。例えば、サインインして MFA コードを入力したユーザーは 2 つのリクエストとしてカウントされます。許可コード付与でのトークンの引き換えには、UserAuthentication カテゴリのクォータと同じレートで追加のクォータ割り当てが適用されます。

³ このカテゴリの個々のオペレーションには、単一のユーザープールに対して 5 RPS を超えるレートでオペレーションが呼び出されないようにする制約があります。

ユーザープールドメインの一括リクエストレート制限

以下のクォータは、ユーザープールドメインへのリクエストの総数に適用されます。

運用	説明	デフォルトクォータ (RPS)	引き上げ可能
送信元 IP からのリクエスト	1 つの IP アドレスから 1 つのドメインへのリクエスト数	300	いいえ

運用	説明	デフォルトクォータ (RPS)	引き上げ可能
アプリケーションクライアントへのリクエスト	1 つのドメイン内の 1 つのアプリケーションクライアント ID に対するリクエスト数	300	いいえ
ドメインへのリクエスト	1 つのユーザープールドメインのサービスに対するリクエストの総数	500	いいえ
JSON ウェブキードキュメントのリクエスト	1 対 1 jwks.json での リクエストの量 AWS アカウント AWS リージョン	50,000	いいえ

Amazon Cognito ID プール (フェデレーティッドアイデンティティ) API オペレーションリクエストレートのクォータ

運用	説明	デフォルトクォータ (RPS) ¹	引き上げ可能	クォータ引き上げの対象
GetId	ID プールからアイデンティティ ID を取得します。	25	はい	アカウントチームにお問い合わせください。
GetOpenIdToken	クラシックワークフローの ID プールから OpenID トークンを取得します。	200	はい	アカウントチームにお問い合わせください。

運用	説明	デフォルト クォータ (RPS) ¹	引き上げ可能	クォータ引き上げの対象
GetCredentialsForIdentity	拡張ワークフローの ID プールから AWS 認証情報を取得します。	200	はい	アカウントチームにお問い合わせください。
GetOpenIdTokenForDeveloperIdentity	デベロッパーワークフローの ID プールから OpenID トークンを取得します。	50	はい	アカウントチームにお問い合わせください。
ListIdentities	アイデンティティプール内の ID のリストを取得します。	5	はい	アカウントチームにお問い合わせください。
DeleteIdentities	アイデンティティプールから登録した ID を 1 つ以上削除します。	10	[Yes (はい)]	アカウントチームにお問い合わせください。
TagResource	アイデンティティプールにタグを適用します。	5	はい	アカウントチームにお問い合わせください。
UntagResource	アイデンティティプールからタグを削除します。	5	はい	アカウントチームにお問い合わせください。

運用	説明	デフォルト クォータ (RPS) ¹	引き上げ可能	クォータ引き上げの対象
ListTagsForResource	アイデンティティプールに適用されたタグのリストを表示します。	10	[Yes (はい)]	アカウントチームにお問い合わせください。

¹ デフォルトのクォータは、AWS リージョン 内の任意の ID プールの最小リクエストレートクォータです AWS アカウント。リージョンによっては RPS クォータが高くなる場合があります。

リソースの番号とサイズのクォータ

リソースクォータは、Amazon Cognito のリソース、入力フィールド、期間、およびその他の機能の最大数またはサイズです。

一部のリソースクォータの調整は、サービスクォータコンソールまたは[サービス制限の引き上げフォーム](#)からリクエストできます。Service Quotas コンソールからクォータの引き上げをリクエストするには、Service Quotas ユーザーガイドの「[Requesting a quota increase](#)」を参照してください。Service Quotas でクォータが利用できない場合は、[Service limit increase フォーム](#)を使用してください。

Note

リージョンあたりのユーザープールなどの AWS アカウント レベルのリソースクォータは、それぞれの Amazon Cognito リソースに適用されます AWS リージョン。例えば、米国東部 (バージニア北部) に 1,000 個のユーザープールを持ち、欧州 (ストックホルム) に別の 1,000 個のユーザープールを持つことができます。

次の表は、デフォルトのリソースクォータと、それらが調整可能かどうかを示しています。

Amazon Cognito ユーザープールのリソースクォータ

以下のクォータは、ユーザープールに作成できる項目の最大数または最大長を示しています。

[リソース]	クォータ	引き上げ可能	最大クォータ
ユーザープールあたりのアプリケーションクライアント	1,000	はい	10,000
リージョンごとのユーザープール	1,000	はい	10,000
ユーザープールあたりのアイデンティティプロバイダー	300	はい	1,000
ユーザープールあたりのリソースサーバー	25	はい	300
ユーザープールあたりのユーザー数	40,000,000	はい	アカウントチームにお問い合わせください。
トークン生成前の Lambda トリガーの合計変更数 ¹	5,000	はい	アカウントチームにお問い合わせください。
ユーザープールあたりのマネージドログインのブランディングスタイル数	20	いいえ	該当なし
ユーザープールあたりのマネージドログイン用語ドキュメント	40	いいえ	該当なし
ユーザープールあたりのカスタム属性数	50	いいえ	該当なし

[リソース]	クォータ	引き上げ可能	最大クォータ
属性あたりの文字数	2,048 バイト	いいえ	該当なし
カスタム属性名の文字数	20	いいえ	該当なし
パスワードポリシーに最低限必要なパスワード文字数	6 ~ 99	いいえ	該当なし
ごとに毎日送信される E メールメッセージ AWS アカウント ²	50	いいえ	該当なし
リクエスト IP アドレスごとに 1 つの E メールアドレスに 1 時間ごとに送信される E メール MFA メッセージ数	5-20	いいえ	該当なし
E メール件名の文字数	140	いいえ	該当なし
E メールメッセージの文字数	20,000	いいえ	該当なし
SMS 検証メッセージの文字数	140	いいえ	該当なし
パスワードの文字数	256	いいえ	該当なし
ID プロバイダー名の文字数	32	いいえ	該当なし
SAML レスポンスの文字数	100,000	いいえ	該当なし

[リソース]	クォータ	引き上げ可能	最大クォータ
ID プロバイダーあたり識別子数	50	いいえ	該当なし
ユーザーにリンクされた ID 数	5	いいえ	該当なし
ユーザーあたりのパスキー/WebAuthn 認証アプリケーション	20	いいえ	該当なし
アプリケーションクライアントあたりのコールバック URL 数	100	いいえ	該当なし
アプリケーションクライアントあたりのログアウト URL 数	100	いいえ	該当なし
リソースサーバーあたりのスコープ	100	いいえ	該当なし
アプリケーションクライアントあたりのスコープ数	50	いいえ	該当なし
リージョンあたりのカスタムドメイン数	4	いいえ	該当なし
各ユーザーが所属できるグループ数	100	いいえ	該当なし
ユーザープールあたりのグループ	10,000	いいえ	該当なし

¹ このクォータは、[トークン生成前の Lambda トリガー](#) からのトークンで発生する場合があります。トランザクション 1 つにおけるアクセストークンと ID トークンの既存および追加のクレームと

スコープの数の合計は、このクォータ以下になる必要があります。抑制されたクレームとスコープは、このクォータには加算されません。

² このクォータは、Amazon Cognito ユーザープールでデフォルトの E メール機能を使用している場合にのみ適用されます。E メールの配信ボリュームがより高い場合は、Amazon SES の E メール設定を使用するようにユーザープールを設定します。この制限は毎日 09:00 UTC にリセットされます。詳細については、「[Amazon Cognito ユーザープールの E メール設定](#)」を参照してください。

Amazon Cognito ユーザープールセッションの有効性パラメータ

以下のクォータは、ユーザープールの認証アーティファクトとユーザーセッションの期間中に使用可能な設定を示しています。

トークン	クォータ
ID トークン	5 分 ~ 1 日
更新トークン	1 時間 ~ 3,650 日
アクセストークン	5 分 ~ 1 日
ホストされた UI セッション cookie	1 時間
認証セッショントークン	3 分 ~ 15 分

Amazon Cognito ユーザープールコードセキュリティリソースクォータ (調整不可)

以下のクォータは、サインイン、サインアップ、パスワードリセットのコードに関連する利用可能な期間を示しています。

[リソース]	クォータ
サインアップ確認コードの有効期間	24 時間
ユーザー属性検証コードの有効期間	24 時間
多要素認証 (MFA) コードの有効期間	3 ~ 15 分

[リソース]	クォータ
パスワードを忘れた場合のコードの有効期間	1 時間
ユーザーあたりの ConfirmForgotPassword および ForgotPassword リクエストの最大数/時間 ¹	5 ~ 20
ユーザーあたりの ResendConfirmation Code リクエストの最大数/時間	5
ユーザーあたりの ConfirmSignUp リクエストの最大数/時間	15
ユーザーあたりの ChangePassword リクエストの最大数/時間	5
ユーザーあたりの GetUserAttributeVerificationCode リクエストの最大数/時間	5
ユーザーあたりの VerifyUserAttribute リクエストの最大数/時間	15

¹ Amazon Cognito は、パスワード更新リクエストのリスク要因を評価し、評価したリスクレベルに応じたクォータを割り当てます。詳細については、「[パスワードを忘れた場合の対応](#)」を参照してください。

Amazon Cognito ユーザープールへのユーザーインポートジョブのリソースクォータ

以下のクォータは、ユーザーインポートジョブに使用できるリソースと制限を示しています。

[リソース]	クォータ	引き上げ可能	最大クォータ
ユーザープールあたりのユーザーインポートジョブ	1,000	はい	アカウントチームにお問い合わせください。

[リソース]	クォータ	引き上げ可能	最大クォータ
ユーザーインポート CSV 行あたりの最大文字数	16,000	いいえ	該当なし
最大 CSV ファイルサイズ	100 MB	いいえ	該当なし
CSV ファイルあたりのユーザーの最大数	500,000	いいえ	該当なし

Amazon Cognito アイデンティティプール (フェデレーテッドアイデンティティ) のリソースクォータ

以下のクォータは、アイデンティティプールに作成できる項目の最大数または最大長を示しています。

[リソース]	クォータ	引き上げ可能	最大クォータ
アカウントあたりのアイデンティティプール	1,000	はい	該当なし
ID プールあたりの Amazon Cognito ユーザープールプロバイダー数	50	はい	1,000
ID プール名の文字長	128 バイト	いいえ	該当なし
ログインプロバイダー名の文字長	2,048 バイト	いいえ	該当なし
ID プールあたりのアイデンティティ数	無制限	いいえ	該当なし

[リソース]	クォータ	引き上げ可能	最大クォータ
ロールマッピングを指定できる ID プロバイダー数	10	いいえ	該当なし
単一のリストまたは lookup コールからの結果の数	60	いいえ	該当なし
ロールベースのアクセス制御 (RBAC) ルール	25	いいえ	該当なし

Amazon Cognito Sync リソースクォータ

以下のクォータは、Amazon Cognito Sync で作成できる項目の最大数または最大長を示しています。

[リソース]	クォータ	引き上げ可能	最大クォータ
アイデンティティあたりのデータセット	20	はい	アカウントチームにお問い合わせください。
データセットあたりのレコード	1,024	はい	アカウントチームにお問い合わせください。
1 つのデータセットのサイズ	1 MB	はい	アカウントチームにお問い合わせください。
データセット名の文字数	128 バイト	いいえ	該当なし

[リソース]	クォータ	引き上げ可能	最大クォータ
正常なリクエスト後の一括発行の待機時間	24 時間	いいえ	該当なし

Amazon Cognito のドキュメント履歴

次の表は、Amazon Cognito のドキュメントへの重要な追加項目を示しています。また、お客様からいただいたフィードバックを反映して、ドキュメントの小さな更新を頻繁に行っています。フィードバックを送信するには、Amazon Cognito ドキュメントの任意のページで下部にあるフィードバックリンクを使用してください。

変更	説明	日付
アプリケーションクライアントのクライアントシークレットローテーション。	シークレットローテーションのために、アプリケーションクライアントごとに最大 2 つのシークレットを関連付けることができるようになりました。シークレットを追加するときに、独自のカスタムシークレット値を指定することもできます。	2026 年 1 月 30 日
インバウンドフェデレーション Lambda トリガーは、Amazon Cognito ユーザープールで使用できます。	インバウンドフェデレーション Lambda トリガーを使用して、外部 ID プロバイダーによる認証プロセス中にフェデレーションユーザー属性を交換できるようになりました。	2026 年 1 月 28 日
AWS PrivateLink は Amazon Cognito ID プールで使用できます。	VPC エンドポイントを介して Amazon Cognito ID プール API オペレーションにリクエストを送信できるようになりました。	2025 年 12 月 10 日
AWS PrivateLink は Amazon Cognito ユーザープールで使用できます。	VPC エンドポイントを介して Amazon Cognito ユーザープール API オペレーションにリクエストを送信できるようになりました。	2025 年 11 月 4 日

[Amazon Cognito ユーザープールがリソースバインディングをサポートするようになりました。](#)

リソースサーバーは、RFC 8707 リソースバインディングにより、アクセストークンのオーディエンス検証を実行できるようになりました。

2025 年 10 月 24 日

[トラブルシューティングの章を追加しました](#)

ユーザープールの一般的な問題のソリューションに関する新しい章。

2025 年 10 月 24 日

[ユーザープールの JSON ウェブキードキュメントのリクエストに新しいクォータを追加しました。](#)

ユーザープール OAuth 2.0 認可サーバーから `.well-known/jwks.json` を取得するための新しいクォータエントリ。

2025 年 10 月 13 日

[マネージドログインの利用規約ドキュメント。](#)

マネージドログインのサインアップページに利用規約とプライバシーポリシーへのリンクを追加できるようになりました。

2025 年 10 月 2 日

[アイデンティティプールの新しい使用方法を示すサンプルアプリケーション。](#)

アイデンティティプールの機能を紹介するサンプルアプリケーションを設定できるようになりました。

2025 年 9 月 30 日

[アイデンティティプールの IAM ロール信頼ポリシーの要件を更新しました。](#)

OIDC (ウェブ ID) フェデレーションのオーディエンスとして条件キーが指定する範囲が 1 つ以上のアイデンティティプールである場合限り、Amazon Cognito アイデンティティプールのサービスプリンシパルのロール信頼ポリシーを変更できるようになりました。

2025 年 8 月 1 日

[Amazon Cognito がメキシコ \(中部\) で利用可能になりました AWS リージョン。](#)

メキシコ (中部) リージョンで Amazon Cognito リソースを作成できるようになりました。

2025 年 7 月 24 日

[Amazon Cognito がアジアパシフィック \(タイ\) で利用可能になりました AWS リージョン。](#)

アジアパシフィック (タイ) リージョンで Amazon Cognito リソースを作成できるようになりました。

2025 年 7 月 24 日

[AWS WAF マネージドログイン ACLs。](#)

マネージドログインブランドバージョンを持つユーザープールアプリケーションに AWS WAF ウェブ ACL ルールを適用できるようになりました。

2025 年 6 月 24 日

[Lambda トリガーの例を更新しました。](#)

カスタム E メールと SMS 送信者 Lambda トリガーのサンプル関数を更新して、Node.js 22.x との互換性を持たせました。サンプルは、アクセスしてテストすることも容易になりました。

2025 年 5 月 19 日

[新しい prompt パラメータ。](#)

prompt パラメータを使用して、既存のマネージドログインセッションの再認証をより詳細に制御できるようになりました。このパラメータの値をサードパーティプロバイダーに渡すこともできます。

2025 年 5 月 15 日

<u>M2M リクエストのクライアントメタデータ。</u>	クライアント認証情報、または Machine to Machine (M2M) リクエストで、クライアントメタデータを渡せるようになりました。Amazon Cognito は M2M クライアントメタデータをトークン生成前の Lambda トリガーに渡します。	2025 年 4 月 29 日
<u>更新トークンのローテーション。</u>	更新トークンリクエストで、新しい更新トークンを取得し、元のトークンを無効化できるようにになりました。	2025 年 4 月 22 日
<u>Amazon Cognito がアジアパシフィック (マレーシア) で利用可能になりました AWS リージョン。</u>	アジアパシフィック (マレーシア) リージョンで Amazon Cognito リソースを作成できるようになりました。	2025 年 3 月 7 日
<u>マシン ID のアクセストークンのカスタマイズ。</u>	トークン生成前の Lambda トリガーにバージョン 3 イベントを追加しました。このイベントでは、Machine to Machine (M2M) 認可のクライアント認証情報付与でアクセストークンのクレームとスコープを変更します。	2025 年 3 月 3 日
<u>AmazonCognitoPowerUser AWS マネージドポリシーに関する情報を更新しました。</u>	Amazon Cognito ユーザープールパワーユーザーの AWS マネージドポリシーに AWS End User Messaging SMS オペレーションを追加しました。	2025 年 2 月 27 日

[OpenID Connect \(OIDC\) 統合の概要を更新しました。](#)

Amazon Cognito が OIDC ID プロバイダーを使用して認証する方法を示す図を追加しました。

2025 年 2 月 25 日

[MFA ロジックに関する情報を追加しました。](#)

Amazon Cognito がランタイムにユーザープール多要素認証 (MFA) 設定をユーザーに適用する方法を示す図を追加しました。

2025 年 2 月 25 日

[Amazon Cognito ユーザープールのセキュリティのベストプラクティスを追加しました。](#)

シークレットを保護し、セキュリティのベストプラクティスに従うことに関するページをユーザープール設定に追加しました。

2025 年 2 月 25 日

[ユーザープールの開始方法リソースの更新。](#)

Amazon Cognito ユーザープールの使用開始エクスペリエンスに、新しいコンソール設計とアプリケーションオプションが追加されました。

2024 年 11 月 21 日

[機能プランの新しい料金モデル。](#)

ユーザープールの請求モデルを更新しました。高度なセキュリティ機能は脅威保護に変更されました。高度なセキュリティ機能ライセンスのコンポーネントが、エッセンシャル機能プランとプラス機能プランに含まれるようになりました。

2024 年 11 月 21 日

[新しいマネージドログイン機能。](#)

マネージドログインを開始しました。これは、ホストされた UI の更新です。

2024 年 11 月 21 日

[新しい認証方法と新しい認証フロー。](#)

パスキーとワンタイムパスワードを使用して Amazon Cognito ユーザープールにサインインできるようになりました。

2024 年 11 月 21 日

[AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy に関する情報を更新しました。](#)

認証されていない ID のスコープダウンに関する AWS 管理ポリシーの AWS Key Management Service オペレーションをインラインポリシーから AWS 管理ポリシーに移動しました。

2024 年 11 月 1 日

[login_hint パラメータが追加されました。](#)

ホストされた UI、OIDC IdP および Google IdP の認可リクエストにユーザー名ヒントを追加できるようになりました。

2024 年 10 月 3 日

[E メール MFA 用の新しい高度なセキュリティ機能。](#)

高度なセキュリティ機能を備えた E メールメッセージで多要素認証 (MFA) コードを送信できるようになりました。

2024 年 9 月 12 日

[新しいコンテンツとページの変更。](#)

タイトルの変更、不要なコンテンツの削除、シナリオベースの概要の追加、ユーザープール OIDC とホストされた UI エンドポイントの記載のユーザープールセクションへの移動を行いました。

2024 年 9 月 9 日

[AmazonCognitoUnAuthenticatedIdentitiesSessionPolicy](#) に関する情報を更新しました。

ID プール内の認証されていない ID のスコープダウンのための AWS マネージドポリシーで、Amazon Location Service が許可されるようになりました。

2024 年 8 月 9 日

[Lambda トリガーによるカスタム認証用の新しい脅威防止と、脅威検出の強化。](#)

脅威保護を使用してカスタム認証サインインを分析し、アダプティブ認証レスポンスを適用できるようになりました。また、脅威保護では、試行間の地理的距離がありえないかどうかについて、サインイントラフィックを分析するようにもなりました。

2024 年 8 月 8 日

[パスワードの再利用防止とユーザーアクティビティログのエクスポートのための新しい高度なセキュリティ機能。](#)

ユーザーアクティビティログをエクスポートし、Amazon Cognito ユーザープールに高度なセキュリティ機能を持つパスワード履歴ポリシーを設定できるようになりました。

2024 年 8 月 6 日

[Amazon Cognito がカナダ西部 \(カルガリー\) およびアジアパシフィック \(香港\) で利用可能になりました AWS リージョン。](#)

カナダ西部 (カルガリー) とアジアパシフィック (香港) のリージョンで Amazon Cognito リソースを作成できるようになりました。

2024 年 7 月 9 日

[高度なセキュリティのためのアプリケーション動作の説明を改善](#)

高度なセキュリティアダプティブ認証のためのデバイスコンテキストデータに関する情報を更新しました。

2024 年 6 月 10 日

トークン生成前 Lambda トリガーで複雑なオブジェクトのサポートを追加しました	ID トークンとアクセストークンのクレームに配列と JSON オブジェクトを追加できるようになりました。	2024 年 5 月 30 日
Verified Permissions と Amazon Cognito に関する情報を更新しました。	Amazon Verified Permissions と Amazon Cognito との統合がさらに直接的になりました。	2024 年 5 月 15 日
マルチリージョン Amazon SES 検証済み ID。	Amazon SES AWS リージョン を使用しない場合、Amazon Cognito ユーザープールは 2 つのリモートリージョン間で E メールをロードバランシングします。	2024 年 5 月 10 日
M2M の認可とコスト管理に関する情報を追加しました。	Amazon Cognito ユーザープールで Machine to Machine (M2M) ユースケースにクライアント認証情報付与を使用する方法について説明します。	2024 年 5 月 9 日
Amazon Cognito が欧州 (スペイン) およびアジアパシフィック (ハイデラバード) で利用可能になりました AWS リージョン。	欧州 (スペイン) とアジアパシフィック (ハイデラバード) のリージョンで Amazon Cognito リソースを作成できるようになりました。	2024 年 4 月 15 日
Amazon Cognito がアジアパシフィック (メルボルン) で利用可能になりました AWS リージョン。	アジアパシフィック (メルボルン) リージョンで Amazon Cognito リソースを作成できるようになりました。	2024 年 4 月 4 日

[Amazon Cognito ユーザープ
ール用の Flutter の Android アプ
リケーションの例を追加しま
した。](#)

Amazon Cognito 用のスター
ターモバイルアプリケーション
は、GitHub の Flutter アプ
リケーションの例から構築で
きます。

2024 年 4 月 4 日

[新しい入門コンテンツ](#)

入門、一般的シナリオ、マル
チテナントのベストプラク
ティス、サインイン後のリ
ソースへのアクセスに関する
コンテンツを拡張しました。

2024 年 4 月 1 日

[Amazon Cognito が欧州
\(チューリッヒ\) で利用可能
になりました AWS リージョ
ン。](#)

欧州 (チューリッヒ) リージョ
ンで Amazon Cognito リソー
スを作成できるようになりま
した。

2024 年 3 月 14 日

[Amazon Cognito が中東 \(UAE\)
で利用可能になりました AWS
リージョン。](#)

中東 (UAE) リージョンで
Amazon Cognito リソースを作
成できるようになりました。

2024 年 3 月 8 日

[新しい SAML 機能とコンテン
ツの改善。](#)

SAML リクエストへの署名
、SAML レスポンスの暗号化
、IdP が開始した SAML SSO
のセットアップができるよう
になりました。

2024 年 2 月 1 日

[クォータの引き上げが可能に
なりました。](#)

Amazon Cognito リクエスト
レートクォータの追加容量を
購入できるようになりまし
た。

2024 年 1 月 25 日

[Amazon Cognito アイデンティティプールは、Service Quotas のリクエストレートをサポートしています。](#)

Amazon Cognito アイデンティティプールの 1 秒あたりのリクエスト数 (RPS) クォータをモニタリングし、Service Quotas コンソールで引き上げをリクエストできるようになりました。

2023 年 12 月 19 日

[アクセストークンの内容をカスタマイズするための新しい機能を追加しました。](#)

ユーザープールアクセストークンのクレームとスコープを追加、変更、削除できるようになりました。

2023 年 12 月 12 日

[アプリケーションクライアントと OAuth スコープに関するコンテンツを改善しました。](#)

[アプリケーションクライアントによるアプリケーション固有の設定とスコープ、M2M、リソースサーバー](#)をわかりやすくするための編集と訂正。従来のコンソールの説明を削除しました。

2023 年 11 月 14 日

[デバイスとデバイス認証に関するコンテンツを改善しました。](#)

デバイスキーとデバイス SRP による認証の使用に関する新しいコンテンツ。

2023 年 10 月 18 日

[AWS マネジメントコンソールガイダンスを更新しました。](#)

ユーザープールコンソールリファレンスを削除し、関連するテーマ内のトピックを並べ替え、Amazon Cognito コンソールのタブベースの整理に関するガイダンスを追加しました。

2023 年 8 月 30 日

LOGIN エンドポイントへの直接アクセスの強調を解除しました。	ユーザープール ログインエンドポイント の視覚的概要を追加し、 認可エンドポイント を使って認証を開始することを強調しました。	2023 年 8 月 30 日
Amazon Cognito がアジアパシフィック (大阪) とイスラエル (テルアビブ) で利用可能になりました AWS リージョン。	アジアパシフィック (大阪) リージョンとイスラエル (テルアビブ) リージョンで Amazon Cognito リソースを作成できるようになりました。	2023 年 8 月 30 日
Amazon Verified Permissions を使った Amazon Cognito の認可に関する情報が追加されました。	アプリ内で検証済みアクセス許可 API を呼び出し、中央機関によるアクセス決定を行うことができます。	2023 年 8 月 1 日
ユーザープールの詳細なユーザーアクティビティを Amazon CloudWatch Logs に記録する新機能を追加しました。	E メールと SMS メッセージの配信エラーを、CloudWatch ロググループに記録できるようになりました。	2023 年 8 月 1 日
ID プールゲストユーザーの AWS マネージドポリシーに関する情報を更新しました。	ID プールゲストユーザーのアクセス許可のスコープダウンに、インラインセッションポリシーと AWS マネージドセッションポリシーの両方が含まれるようになりました。	2023 年 5 月 16 日
Amazon Cognito アイデンティティプールに関するコンテンツの改善と新しいコンソールの説明。	新しいコンソールのチュートリアルを追加し、新しいコンソールエクスペリエンスを反映しました。また、アイデンティティプールに関するコード統合の詳細を改善しました。	2023 年 5 月 16 日

サービスホームページとユーザープールホームページの追加と改善。	Amazon Cognito と ユーザープール の概要ページを更新しました。	2023 年 5 月 16 日
ユーザープールトークンのドキュメントが全般的に改善されました。	トークンの例を更新し、トークンの検証に関する新しい情報を追加しました。	2023 年 2 月 16 日
Amazon Cognito アイデンティティプールのデータイベントを AWS CloudTrail でログに記録できるようになりました。	CloudTrail は、データイベントをログに記録するトレイル内において、大量の API オペレーションの Amazon Cognito ID プールの選択をサポートします。	2023 年 2 月 15 日
Lambda トリガーの例と説明が更新されました。	Lambda トリガーの例は JavaScript バージョン 3 に更新されました。Lambda トリガーを API アクションに直接関連させることができるようになりました。	2023 年 1 月 31 日
Amazon Cognito ID プールは、認証されていないセッションに AWS 管理ポリシーを適用します。	拡張フローを使用して認証する ID プールユーザーには、セッションに追加の AWS 管理ポリシーが適用されるようになりました。	2023 年 1 月 31 日
コードの例を追加しました。	このガイドには、さまざまなプログラミング言語の Amazon Cognito アプリのサンプルコードが含まれるようになりました。	2023 年 1 月 23 日
Amazon Cognito ユーザープールを使用した API モデルと認証に関する情報を追加しました。	Amazon Cognito ユーザープールには、リクエスト承認用の複数の API インターフェイスと形式があります。	2022 年 12 月 15 日

<u>Amazon Cognito が欧州 (ミラノ) で利用可能になりました AWS リージョン。</u>	欧州 (ミラノ) リージョンで Amazon Cognito ユーザープールを作成できるようになりました。	2022 年 12 月 6 日
<u>ユーザープール削除保護に関する情報を追加しました。</u>	を使用して新しいユーザープールを作成すると AWS マネジメントコンソール、デフォルトで削除から保護されるようになりました。	2022 年 10 月 20 日
<u>ホストされた UI のユーザーガイドと、ホストされた UI の TOTP MFA に関する情報を追加しました。</u>	ユーザーは、Amazon Cognito がホストする UI で TOTP MFA デバイスを登録できるようになりました。デフォルトのホストされた UI をプレビューできるようになりました。	2022 年 9 月 8 日
<u>AWS WAF と Amazon Cognito に関する情報を追加しました。</u>	AWS WAF ウェブ ACL を Amazon Cognito ユーザープールに関連付けることができるようになりました。	2022 年 8 月 3 日
<u>AWS CloudTrail イベントの例を追加しました。</u>	Amazon Cognito は、フェデレーションとホストされた UI リクエストを証跡に記録するようになりました	2022 年 6 月 15 日
<u>2 段階の属性検証に関する情報を追加しました。</u>	ユーザーがログインする前に新しいメールアドレスまたは電話番号を確認する必要があるかどうかを選択できるようになりました。	2022 年 6 月 9 日

[フェデレーションのドキュメントを更新しました。新しい IP アドレス伝播機能。](#)

ユーザープールのソーシャル IdP を設定するためのチュートリアルを更新しました。フェデレーションユーザープロファイルと属性マッピングに関する情報を追加しました。高度なセキュリティのために、デバイスのフィンガープリントに関する新しい情報を追加しました。

2022 年 5 月 31 日

[ホストされた UI を操作せずにフェデレーションユーザーにサインインする](#)

Amazon Cognito がユーザーをフェデレーティッドサインインに操作なしに誘導するようにアプリケーションをブックマークする方法に関する新しいページを追加しました。

2022 年 5 月 29 日

[Amazon Cognito ユーザープール用のリージョン内 SMS と E メールメッセージ](#)

SMS メッセージに Amazon Simple Notification Service を使用し、ユーザープール AWS リージョン と同じ の E メールメッセージに Amazon Simple Email Service を使用できるようになりました。

2022 年 3 月 14 日

[クォータページの更新](#)

リソースとリクエストレート
のクォータを追加して明確にしました。

2022 年 1 月 10 日

[新しい Amazon Cognito ユーザープールのコンソールエクスペリエンス](#)

更新された Amazon Cognito コンソールでユーザープールを作成および管理するための手順を更新しました。

2021 年 11 月 18 日

RevokeToken API と取り消しエンドポイント	RevokeToken オペレーションを使用して、ユーザーの 更新トークンを取り消す ことができます。	2021 年 6 月 10 日
マルチテナントのベストプラクティス	マルチテナントアプリケーションのベストプラクティスを追加しました。	2021 年 3 月 4 日
アクセスコントロールの属性	Amazon Cognito Identity Pools は、お客様が AWS リソースへのアクセス権をユーザーに付与する方法として、アクセスコントロール (AFAC) の属性を提供します。認可は、ID プロバイダーからのユーザーの属性 (ID プロバイダーは Amazon Cognito とのフェデレーションにこれらを使用します) に基づいて実行することができます。	2021 年 1 月 15 日
カスタム SMS 送信者の Lambda トリガーとカスタム E メール送信者の Lambda トリガー	カスタム SMS 送信者の Lambda トリガーとカスタム E メール送信者の Lambda トリガーは、サードパーティープロバイダーが Lambda 関数コード内からユーザーに E メールと SMS 通知を送信できるようにします。	2020 年 11 月 30 日
Amazon Cognito トークンの更新	アクセストークン、ID トークン、および更新トークンに更新された有効期限情報が追加されました。	2020 年 10 月 29 日

[Amazon Cognito Service Quotas](#)

Service Quotas は、Amazon Cognito カテゴリクォータで使用できます。Service Quotas コンソールを使用して、クォータの使用状況の表示、クォータ引き上げのリクエスト、およびクォータの使用状況を監視する CloudWatch アラームの作成を実行することができます。この変更の一環として、「Amazon Cognito ユーザープールに利用できる CloudWatch メトリクス」セクションが新しい情報を反映するために更新されました。新しいセクション名は「CloudWatch と Service Quotas でのクォータと使用状況の追跡」です。

2020 年 10 月 29 日

[Amazon Cognito クォータのカテゴリ分類](#)

クォータカテゴリは、クォータの使用状況の監視と引き上げのリクエストに役立てることができます。クォータは、一般的なユースケースに基づいてカテゴリに分類されます。

2020 年 8 月 17 日

[US AWS GovCloud でサポートされている Amazon Cognito](#)

Amazon Cognito が AWS GovCloud (米国) リージョンでサポートされるようになりました。

2020 年 5 月 13 日

[Amazon Cognito Pinpoint ドキュメントの更新](#)

新しいサービスリンクロールが追加されました。「Amazon Cognito ユーザープールでの Amazon Pinpoint 分析の使用」にある手順が更新されました。

2020 年 5 月 13 日

[Amazon Cognito 専用のセキュリティに関する新しい章](#)

セキュリティの章は、組織がサービスの組み込みセキュリティと設定可能なセキュリティの両方に関する詳細情報を取得するのに役立ちます AWS。新しい章では、クラウドとクラウド内のセキュリティに関する情報を提供します。

2020 年 4 月 30 日

[Amazon Cognito アイデンティティプールが「Apple でサインイン」をサポート開始](#)

「Apple でサインイン」は、cn-north-1 リージョンを除く、Amazon Cognito が運営されているその他すべてのリージョンで利用できます。

2020 年 4 月 7 日

[Facebook API の新しいバージョン](#)

Facebook API にバージョン選択を追加しました。

2020 年 4 月 3 日

[ユーザー名の大文字と小文字を区別しない更新](#)

ユーザープールを作成する前に、ユーザー名の大文字と小文字を区別しない設定を有効にすることを推奨事項として追加しました。

2020 年 2 月 11 日

[に関する新しい情報 AWS Amplify](#)

AWS Amplify の SDK とライブラリを使用することによって、ウェブアプリケーションまたはモバイルアプリケーションに Amazon Cognito を統合することに関する情報が追加されました。AWS Amplify より前の Amazon Cognito SDK の使用に関する情報は削除されました。

2019 年 11 月 22 日

[ユーザープールトリガーの新しい属性](#)

Amazon Cognito は、ほとんどのユーザープールトリガーの AWS Lambda 関数に渡すイベント情報に `clientMetadata` パラメータを含めるようになりました。このパラメータを使用して、追加のデータでカスタム認証ワークフローを強化できます。

2019 年 10 月 4 日

[更新された制限](#)

ListUsers API アクションのロットリング制限が更新されました。

2019 年 6 月 25 日

[新しい制限](#)

ユーザープールのソフト制限に、ユーザー数の制限が含まれるようになりました。

2019 年 6 月 17 日

[Amazon Cognito ユーザープールのための Amazon SES E メール設定](#)

Amazon Cognito が Amazon SES 設定を使用してユーザーに E メールを送信するように、ユーザープールを設定できます。この設定では、Amazon Cognito が、これ以外の方法では成し得なかった大規模な配信ボリュームで E メールを送信できるようになります。

2019 年 4 月 8 日

[タグ付けのサポート](#)

Amazon Cognito リソースのタグ付けに関する情報を追加しました。

2019 年 3 月 26 日

[カスタムドメインの証明書を変更する](#)

Amazon Cognito でホストされる UI のホストにカスタムドメインを使用する場合は、必要に応じてこのドメインの SSL 証明書を変更できます。

2018 年 12 月 19 日

[新しい制限](#)

各ユーザーが所属できるグループの最大数に新しい制限が追加されました。

2018 年 12 月 14 日

[制限の更新](#)

ユーザープールのソフト制限が更新されました。

2018 年 12 月 11 日

[メールアドレスと電話番号を確認するためのドキュメント更新](#)

ユーザーがアプリにサインアップするときに E メールまたは電話による確認を要求するようにユーザープールを構成する方法に関する情報を追加しました。

2018 年 11 月 20 日

[E メールをテストするためのドキュメント更新](#)

アプリのテスト中に Amazon Cognito からの E メールを開始するためのガイダンスを追加しました。

2018 年 11 月 13 日

[Amazon Cognito の高度なセキュリティ](#)

新しいセキュリティ機能を追加しました。これにより、デベロッパーは、アプリやユーザーを悪意のあるボットから保護し、ユーザーアカウントをセキュリティで保護して認証情報の侵害を防ぎ、サインイン試行の計算されたリスクに基づいてサインインに必要なチャレンジを自動的に調整できます。

2018 年 14 月 6 日

[Amazon Cognito のホストされる UI のカスタムドメイン](#)

デベロッパーが Amazon Cognito ユーザープール内のホストされた UI に独自の完全なカスタムドメインを使用することを可能にします。

2018 年 6 月 4 日

[Amazon Cognito ユーザープールの OIDC ID プロバイダー](#)

Salesforce や Ping Identity などの OpenID Connect (OIDC) ID プロバイダー経由でのユーザープールサインインを追加しました。

2018 年 5 月 17 日

[Amazon Cognito Lambda 移行トリガー](#)

Lambda の移行トリガー機能を対象とするページが追加されました。

2018 年 4 月 8 日

[Amazon Cognito デベロッパーガイドの更新](#)

トップレベルの「Amazon Cognito とは」と「Amazon Cognito の使用開始方法」を追加しました。また、一般的なシナリオを追加し、ユーザープール TOC を再編成しました。新しい「Amazon Cognito ユーザープールの使用開始方法」セクションを追加しました。

2018 年 4 月 6 日

[Amazon Cognito 高度なセキュリティのベータ](#)

デベロッパーがアプリやユーザーを悪意のあるボットから保護し、インターネットの他の場所で侵害された認証情報に対してユーザーアカウントをセキュリティで保護して、サインインの試行の兆候について計算されたリスクに基づいて、サインインに必要なチャレンジを自動的に調整できるようにする、新しいセキュリティ機能が追加されました。

2017 年 11 月 28 日

[Amazon Pinpoint の統合](#)

Amazon Cognito ユーザープールアプリの分析を提供し、Amazon Pinpoint キャンペーン向けのユーザーデータを強化するために Amazon Pinpoint を使用する機能を追加しました。

2017 年 9 月 26 日

[Amazon Cognito ユーザープールのフェデレーションと組み込みのアプリケーション UI 機能](#)

ユーザーが Facebook、Google、Login with Amazon、または SAML ID プロバイダーを使用してユーザープールにサインインすることを許可する機能を追加しました。カスタマイズ可能な組み込みアプリ UI およびカスタムクレームを使用した OAuth 2.0 のサポートを追加しました。

2017 年 8 月 10 日

[HIPAA および PCI コンプライアンスに関連する機能の変更](#)

ユーザーが電話番号またはメールアドレスユーザー名として使用することを許可する機能が追加されました。

2017 年 7 月 6 日

[ユーザーグループとロールベースのアクセスコントロール機能](#)

ユーザーグループを作成および管理する管理機能が追加されました。管理者は、グループメンバーシップと管理者が作成したルールに基づいて、IAM ロールをユーザーに割り当てることができます。

2016 年 12 月 15 日

[ドキュメントの更新](#)

ユーザープールで AWS Lambda トリガーを使用する方法を示す例を更新しました。

2016 年 11 月 27 日

[ドキュメントの更新](#)

iOS コードサンプルを更新しました。

2016 年 11 月 18 日

[ドキュメントの更新](#)

ユーザーアカウントの認証の流れに関する情報を追加しました。

2016 年 11 月 9 日

ユーザーアカウント作成機能	Amazon Cognito コンソールと API を使用してユーザーアカウントを作成するための管理機能を追加しました。	2016 年 10 月 6 日
ユーザーのインポート機能	Cognito ユーザープール一括インポート機能を追加しました。この機能を使用して、既存の ID プロバイダーから Amazon Cognito ユーザープールにユーザーを移行します。	2016 年 9 月 1 日
Cognito ユーザープールの一般提供	Cognito ユーザープール機能を追加しました。この機能を使用して、ユーザーディレクトリを作成、管理し、ユーザープールを使用してモバイルアプリケーションまたはウェブアプリケーションにサインアップとサインインを追加します。	2016 年 7 月 28 日
SAML サポート	Security Assertion Markup Language 2.0 (SAML 2.0) を通じたアイデンティティプロバイダーによる認証サポートを追加しました。	2016 年 6 月 23 日
CloudTrail の統合	との統合を追加しました AWS CloudTrail。	2016 年 2 月 18 日
イベントの Lambda との統合	Amazon Cognito の重要なイベントに応じて AWS Lambda 関数を実行できます。	2015 年 4 月 9 日
Amazon Kinesis へのデータストリーム	データストリームにコントロールや洞察を提供します。	2015 年 3 月 4 日

OpenID Connect のサポート	OpenID 接続プロバイダーをサポートします。	2014 年 11 月 23 日
プッシュ同期	サイレントプッシュ同期のサポートを有効にします。	2014 年 11 月 6 日
デベロッパーが認証するアイデンティティのサポートの追加	独自の認証およびアイデンティティ管理システムを持つデベロッパーが、Amazon Cognito の ID プロバイダーとして扱われるようになります。	2014 年 9 月 29 日
Amazon Cognito の一般公開開始		2014 年 7 月 10 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。