AWS Cognito

Verified 2026-11-02

Managed sign-up and sign-in for web and mobile apps — user pools, app clients, tokens, triggers, federation, MFA, passkeys and integrations.

Contents

Mental model 2

User pools vs identity pools 2

App clients and sign-in 3

The three tokens 5

Attributes and groups 7

Lambda triggers 8

Federation, MFA and passkeys 10

API Gateway and ALB integration 11

Plans and pricing 13

Further reading 14

Mental model

A **user pool** is an OpenID Connect provider plus a user directory: it signs users in and issues JWTs that your API verifies. An **identity pool** is a different service that exchanges a token for temporary AWS credentials, so a browser can call S3 or DynamoDB directly. Most Cognito confusion is mixing the two — if your backend only needs to know who the user is, you need a user pool and nothing else.

User pools vs identity pools

Need	Use
Sign-up, sign-in, JWTs for your API	User pool
Temporary AWS credentials in a client	Identity pool
Guest access to AWS resources	Identity pool, unauthenticated role

An identity pool trusts one or more providers — a user pool, a social IdP, SAML — and maps each signed-in user to an IAM role. You need it only when the client itself calls AWS APIs. Everything else in this sheet is about user pools.

```
aws cognito-idp create-user-pool \
  --pool-name app-users \
  --username-attributes email \
  --auto-verified-attributes email
```

App clients and sign-in

Option	When
Managed login (hosted pages)	OAuth flows, social and SAML login
SDK `InitiateAuth`	Your own sign-in UI
Public client, no secret	Browser and mobile apps
Confidential client, secret	Server-side apps only

An **app client** is one application's configuration: allowed auth flows, OAuth scopes, callback URLs and token lifetimes. Managed login is the newer hosted UI and requires the Essentials plan or higher; Lite keeps the classic hosted UI. With your own UI, call the user pool API directly.

```
import {
  CognitoIdentityProviderClient as Idp,
  InitiateAuthCommand as Auth,
} from "@aws-sdk/client-cognito-identity-
provider";

const [u, p] = process.argv.slice(2);
const idp = new Idp({ region: "eu-west-1" });
const out = await idp.send(new Auth({
  ClientId: "abc123",
  AuthFlow: "USER_PASSWORD_AUTH",
  AuthParameters: { USERNAME: u, PASSWORD: p },
}));
```

```
const t = out.AuthenticationResult;
console.log(t?.AccessToken, t?.ExpiresIn);
```

> **Gotcha:** If the app client has a secret, every call needs a `SECRET_HASH` parameter. Browser apps cannot keep a secret, so create their client without one.

The three tokens

Token	Proves	Lifetime range
ID token	Who the user is (`aud` = client)	5 min – 1 day, default 1 h
Access token	What it may call (`scope`)	5 min – 1 day
Refresh token	Right to get new tokens	60 min – 10 years, default 30 days

Lifetimes are set per app client. Send the **access token** to APIs: it carries `scope`, `cognito:groups` and `client_id`,

but no `aud`. The ID token carries profile attributes and `aud`. Verify signatures against the pool's JWKS at `https://cognito-idp.<region>.amazonaws.com/<poolId>/.well-known/jwks.json`

```
import { CognitoJwtVerifier }
  from "aws-jwt-verify";

const verifier = CognitoJwtVerifier.create({
  userPoolId: "eu-west-1_AbCdEf",
  tokenUse: "access",
  clientId: "abc123",
});
const token = process.argv[2];
const claims = await verifier.verify(token);
// sub, scope, "cognito:groups", client_id
```

> **Gotcha:** A JWT authorizer that checks `aud` rejects access tokens, which have only `client_id`. API Gateway HTTP APIs fall back to `client_id` when `aud` is absent; hand-rolled verifiers often do not.

Attributes and groups

Item	Quota
Custom attributes per pool	50, not adjustable
Groups per pool	10,000
Groups per user	100
App clients per pool	1,000, adjustable

Custom attributes are named `custom:<name>`. Groups appear in tokens as `cognito:groups` and can carry an IAM role for identity pools. Use groups for coarse roles like `admin`; keep fine-grained permissions in your application.

```
aws cognito-idp admin-add-user-to-group \
  --user-pool-id eu-west-1_AbCdEf \
  --username a@b.c \
  --group-name admin
```

> **Warning:** Custom attributes are permanent. You cannot delete or rename one, or change its type, mutability or length — design the schema before production, or you will be migrating to a new pool.

Lambda triggers

Trigger	Typical use
Pre sign-up	Block domains, auto-confirm
Post confirmation	Create the user's DB row
Pre token generation	Add or remove claims and scopes
Custom message	Brand verification emails

Trigger	Typical use
Define/create/verify auth challenge	Custom auth flows

A trigger is a Lambda function the pool invokes synchronously at a step of the flow; returning an error stops the flow. Pre token generation event version `V1_0` changes only ID tokens; `V2_0` also customizes access tokens and needs the Essentials or Plus plan.

```
export const handler = async (event) => {
  event.response.claimsOverrideDetails = {
    claimsToAddOrOverride: { tenant: "acme" },
  };
  return event;
};
```

> **Gotcha:** `UpdateUserPool` resets every setting you omit, Lambda triggers included. Read the pool with `DescribeUserPool` first and send the full configuration back.

Federation, MFA and passkeys

Feature	Notes
Social IdPs	Google, Facebook, Apple, Amazon
SAML 2.0 and OIDC IdPs	Enterprise SSO
MFA by SMS, TOTP	Any plan
MFA by email	Essentials or higher
Passkeys	Essentials or higher, `USER_AUTH` flow

Federated users sign in through managed login; Cognito maps IdP claims onto pool attributes and issues its own tokens. Passkeys and passwordless one-time codes use

choice-based authentication, the `USER_AUTH` flow, which must be enabled on the app client.

```
aws cognito-idp set-user-pool-mfa-config \
  --user-pool-id eu-west-1_AbCdEf \
  --mfa-configuration OPTIONAL \
  --software-token-mfa-configuration \
    Enabled=true
```

> **Gotcha:** A passkey cannot be the second factor after a password. And one-time-password sign-in is not compatible with required MFA on the pool.

API Gateway and ALB integration

Front end	Mechanism
REST API	Cognito user pool authorizer
HTTP API	JWT authorizer, issuer = pool URL
ALB	`authenticate-cognito` listener action

A REST API authorizer with scopes configured accepts access tokens and checks their scopes. An HTTP API JWT authorizer needs the issuer `https://cognito-idp.<region>.amazonaws.com/<poolId>` and the app client ID as audience. An ALB action works only on HTTPS listeners and forwards claims in the signed `x-amzn-oidc-data` header.

```
{
  "Type": "authenticate-cognito",
  "Order": 1,
  "AuthenticateCognitoConfig": {
    "UserPoolArn": "arn:aws:cognito-idp:...",
    "UserPoolClientId": "abc123",
    "UserPoolDomain": "auth-app"
  }
}
```

> **Warning:** Behind an ALB, verify the signature of `x-amzn-oidc-data` and check that its `signer` is your ALB's ARN. The other `x-amzn-oidc-*` headers are unsigned and spoofable.

Plans and pricing

Plan	Adds
Lite	Basic sign-in, classic hosted UI
Essentials	Managed login, passkeys, email MFA, access-token customization
Plus	Threat protection, compromised-credential checks

New user pools default to Essentials. Billing is per monthly active user (MAU). Lite and Essentials share a free tier of 10,000 MAUs a month for direct and social sign-in; SAML and OIDC federated users get 50 free MAUs; Plus has no free tier. Pools created on or before 22 Nov 2024 keep their old 50,000-MAU free tier.

```
aws cognito-idp create-user-pool \
  --pool-name internal-tools \
  --user-pool-tier LITE
```

> **Tip:** Check the pricing page for per-MAU rates before choosing Plus — it is billed from the first user.

Further reading

- Cognito Developer Guide
- Verifying JSON web tokens
- Quotas in Amazon Cognito
- Amazon Cognito pricing