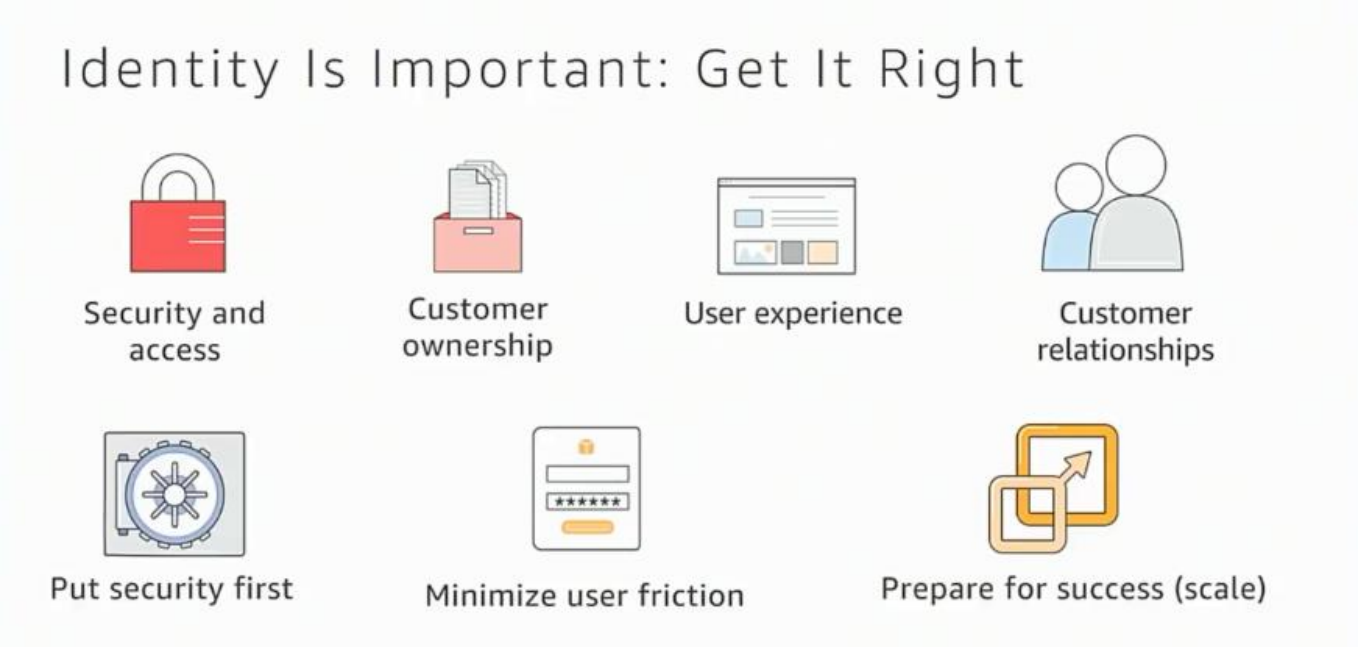
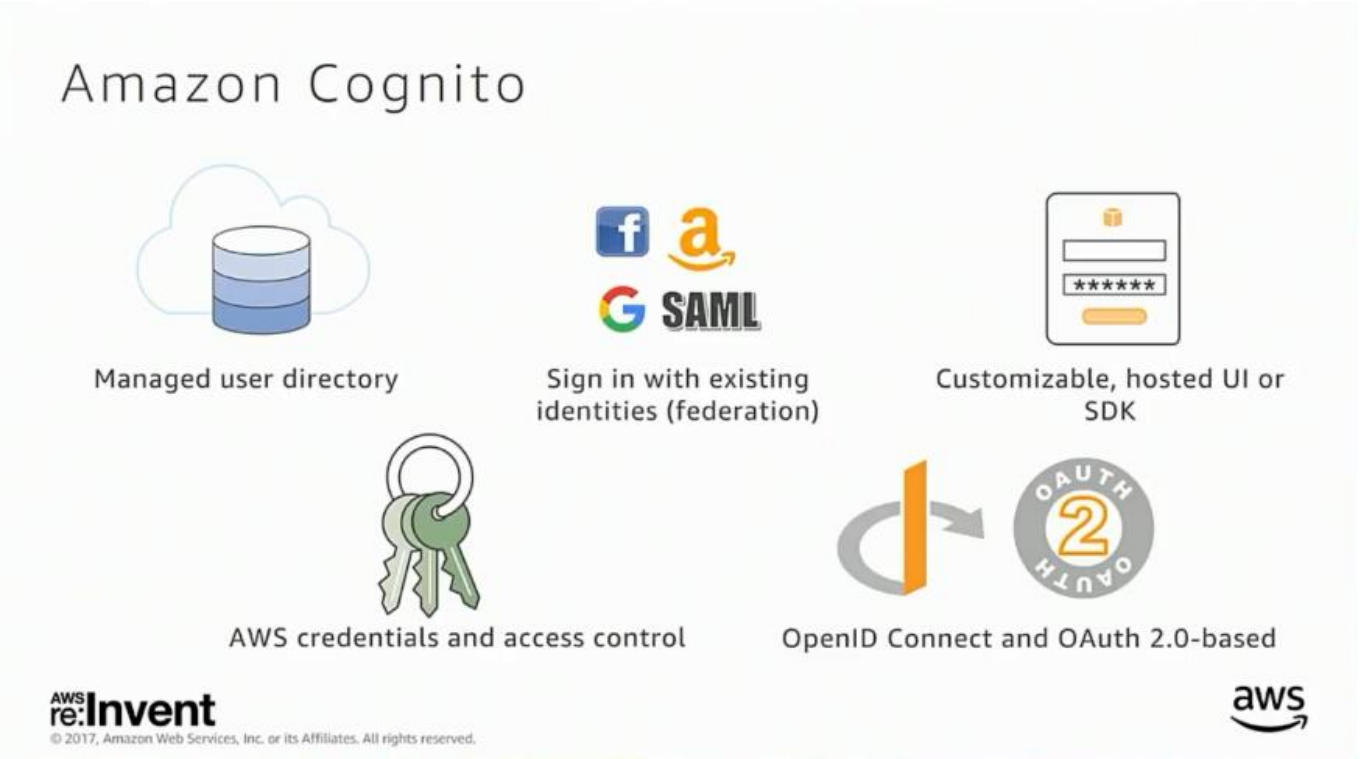
Learn how to set up an end-user directory, secure sign-up and sign-in, manage user profiles, authenticate and authorize your APIs, federate from enterprise and social identity providers, and use ***OAuth*** to integrate with your app—all without any server setup or code. With clear blueprints, we show you how to leverage ***Amazon Cognito*** to administer and secure your end users and enable identity for the applied patterns of mobile, web, and enterprise apps.

What to Expect from This Session

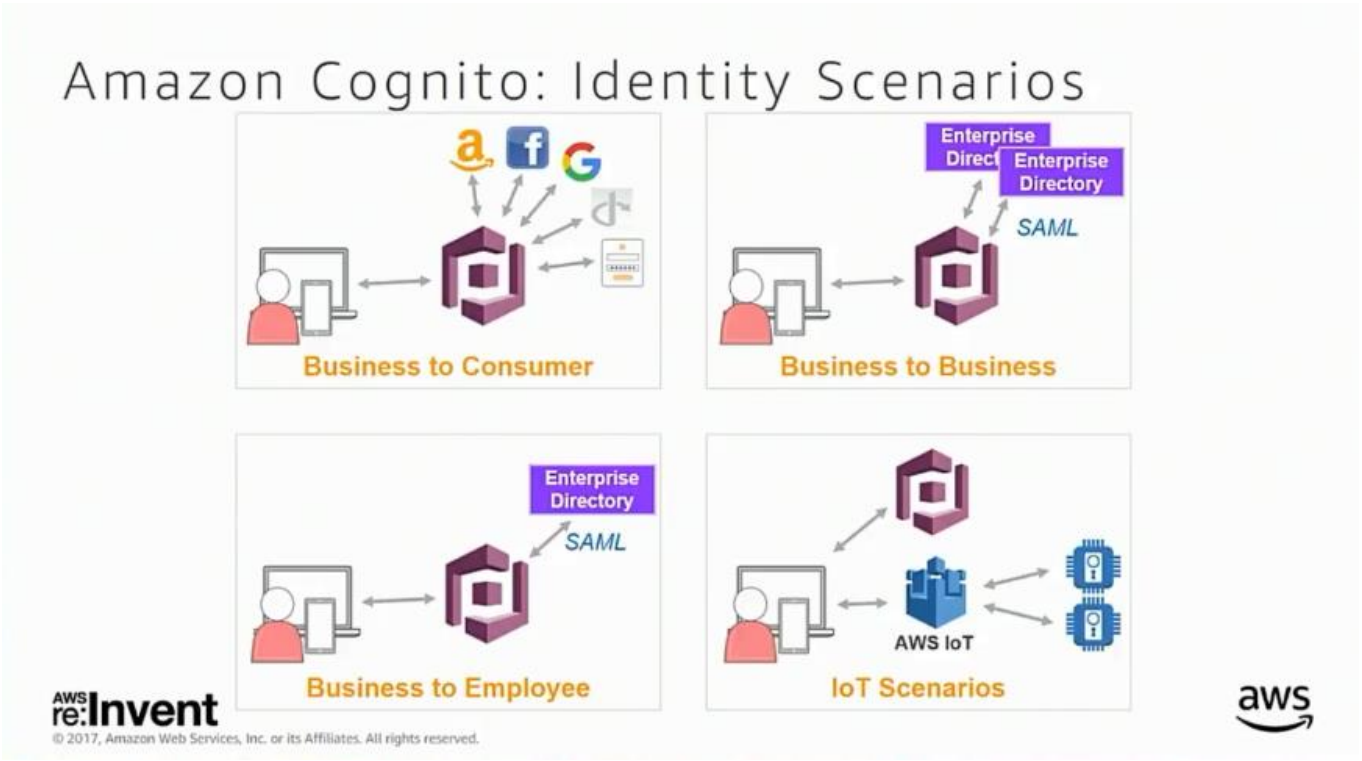
- Amazon Cognito overview
- Use Cases
  - Secure authentication and authorization
  - Simple hosted UI, identity provider federation, and user migration
  - Flexible app integration with custom authentication flows and UIs
- Demo
- Summary and additional resources

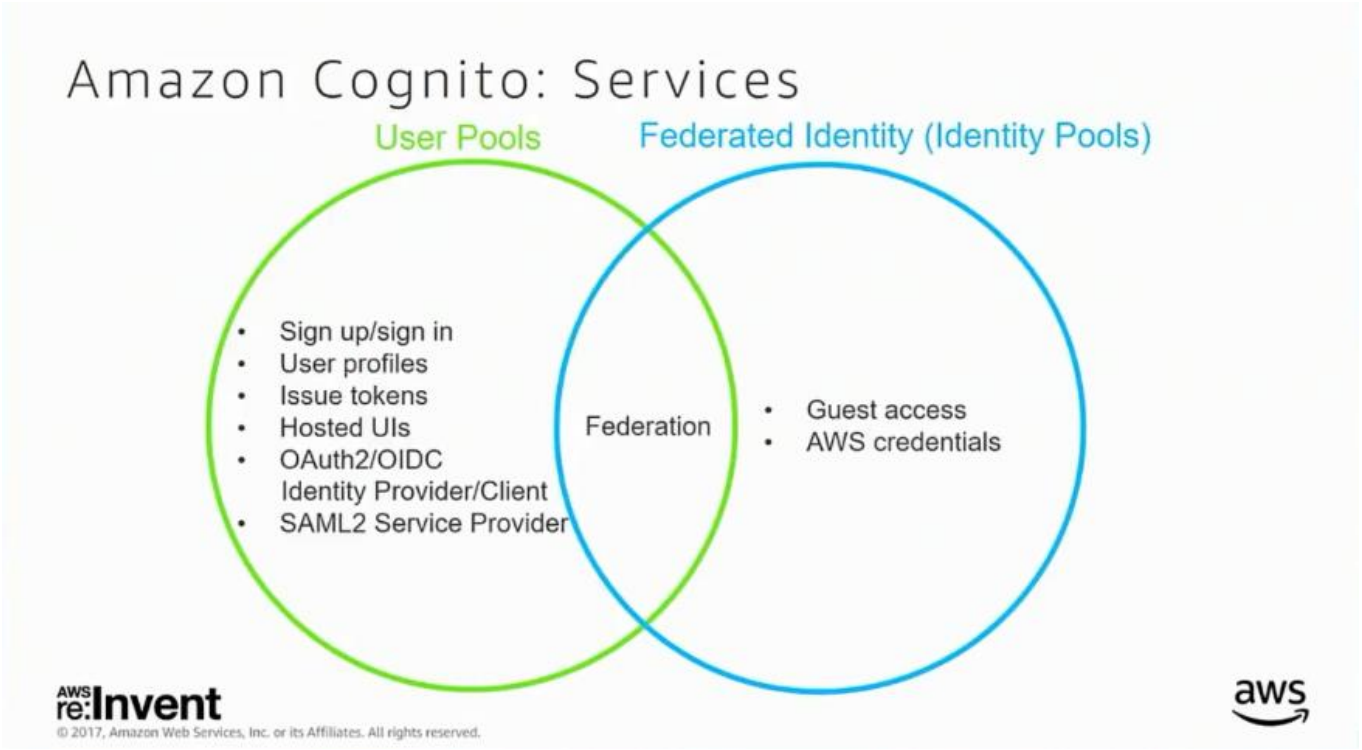


Once you have authenticated your user, you then have a secure channel to access your assets and your backend APIs. Having a user profile enables you to customize the user experience after they get the authentication token after sign in.



AWS can also provide AWS credentials with fine grained Roles for accessing AWS resources like S3 and DynamoDB directly from your app.





AWS Cognito is really 2 services with some overlap. User Pools is the standalone user service.

Use Amazon Cognito User Pools If

You have:

1. Users who want to create an account
2. Users who have an existing social or corporate account

You want:

1. Managed scalable and secure user directory
2. User authentication
3. User profiles
4. OpenID Connect id token, access token, and refresh token to authenticate/authorize against your backend service
5. OAuth 2.0 flows for your app to authenticate with User Pool
6. OAuth 2.0 and SAML2 redirect/POST bindings to authenticate with identity providers
7. Integration with Amazon API Gateway

Use Amazon Cognito Identity Pools If

You have:

1. User that has authenticated with a social or corporate identity provider and has a token
2. User that is unauthenticated

You want:

1. Scoped, time-bound AWS credentials for that identity
2. Direct access to AWS services from your web or mobile app
3. Role mapping

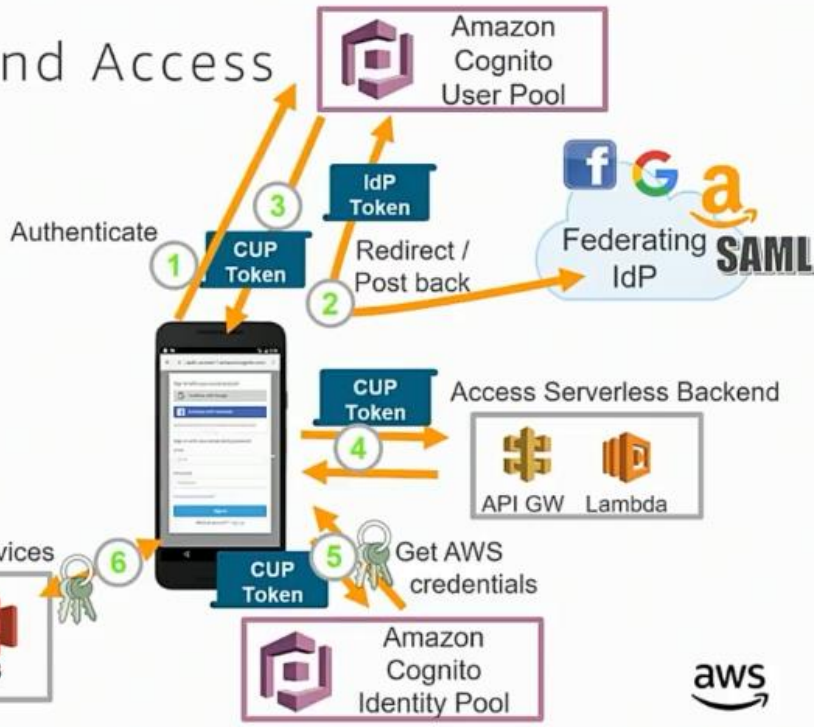
Authentication and Access

Amazon Cognito User Pool

- User pools authenticate users and vend standard tokens
- User pool tokens are used to access backend resources
- Identity pools provide AWS credentials to access AWS services

Authentication and Access

- User pools authenticate users and vend standard tokens
- User pool tokens are used to access backend resources
- Identity pools provide AWS credentials to access AWS services



Secure authentication and authorization

User Pools: Authentication Flow

- Every authentication is one or more rounds of a challenge and a response
- Upon successful completion of the challenges, tokens are issued
- You can customize the authentication flow using an AWS Lambda function

User Pools: Multi-Factor Authentication

- Enable additional factors to verify user identity
  - Send a code via SMS, which the user enters as part of authentication
  - Request a code for a registered time-based software token
- Enhances security
- Enabled at the User Pool level, optionally configured at user level

User Pools: Multi-factor Authentication

Scenario:

- Enhance protection for security-sensitive apps

Recommendation:

- Enable multi-factor authentication on your pool. Choose required or optional
- Configure an IAM role to enable User Pools to send SMS on your behalf via SNS
- Enable phone number verification

Do	Don't
✓ Request increased spending limit for SNS	✗ Forget the end-user experience, i.e., requiring MFA on non-security-sensitive apps

User Pools: Device Tracking

Scenario:

- Reduce friction by avoiding user interaction for known devices

Recommendation:

- Enable Remember Device in AWS Console
- On first sign-in, the Amazon Cognito SDK will store a device identifier and secret securely on the device
- On future sign-ins, Amazon Cognito SDK will automatically authenticate the device
- Amazon Cognito SDK enables you get user consent via callback

Do	Don't
✓ Use AdminListDevices to see users' devices	✗ Forget to enable MFA on your pool

User Pools: Authorization Using Groups

Scenario:

- Authorize access to backend resources and APIs using user groups

Recommendation:

- Add users to User Pool groups using Amazon Cognito console, CLI, or APIs
- Create IAM roles to associate with each User Pool group
- Amazon Cognito id token will contain the following group related claims:
    - cognito:groups has all groups user belongs to
    - cognito:roles has all roles for all those groups
    - cognito:preferred_role has role for group with lowest precedence number
- Groups attributes from federated identity providers (i.e. SAML) can be mapped to User Pool attributes for inclusion in the id token

Do	Don't
✓ Configure unique precedence for groups	✗ Exceed 25 groups

Authorization With User Attributes

Scenario:

- Fine-grained authorization using user attributes and app permissions

Recommendation:

- Use access token and id token attributes for authorizing access to backend APIs and resources
- Define resource server scopes allowed for a User Pool app client
  - E.g. https://myphotosapi.example.com/photos.read
- E.g. Java Spring Security: create Authentication object with custom UserDetails containing id token attributes. Use annotations to authorize APIs:

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
public void adminAPI(...) {
    ....
}
```

```
@PreAuthorize("principal.userType == 'Premium'")
public void premiumFeatureAPI(...) {
    ....
}
```


aws

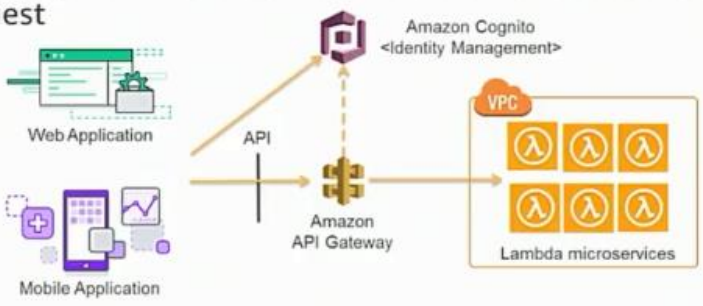
Amazon API Gateway Authorization

Scenario:

- Fine-grained authorization for resources accessed using Amazon API Gateway

Recommendation:

- Configure User Pool authorizer in Amazon API Gateway
- Sign in user to User Pool and get Amazon Cognito tokens
- Send Amazon Cognito id token as HTTP authorization header in Amazon API Gateway request



aws

User Pools: Advanced Security Features Beta

Scenario:

- Protect user profiles from attackers without adding friction

Recommendation:

- Enable advanced security and risk-based Adaptive Multi-Factor Authentication
    - Adds compromised credentials detection
    - Adds challenges to anomalous sign-in attempts

Advanced Security: Compromised Credentials

Detect the reuse of compromised credentials as users sign up, sign in, or change their password

Choose whether to block users from reusing compromised credentials

Advanced Security: Anomaly Detection

Detect anomalies during sign-in events:

- Sign-in from previously unseen or atypical location, from previously unknown device
- Sign-in from IP addresses with a high number of failed sign-in attempts

Customizable

- Set thresholds for blocking requests or requiring MFA depending on the risk level (low, med, high)
- Define whitelist/blacklist IP addresses to exempt from protections or block always

Advanced Security: Reporting

Alert users to suspicious sign-in attempts

View aggregate metrics in AWS CloudWatch on the threats detected by each feature

Review activity see recent sign-in activity for users

User Pools: Signing Out Users

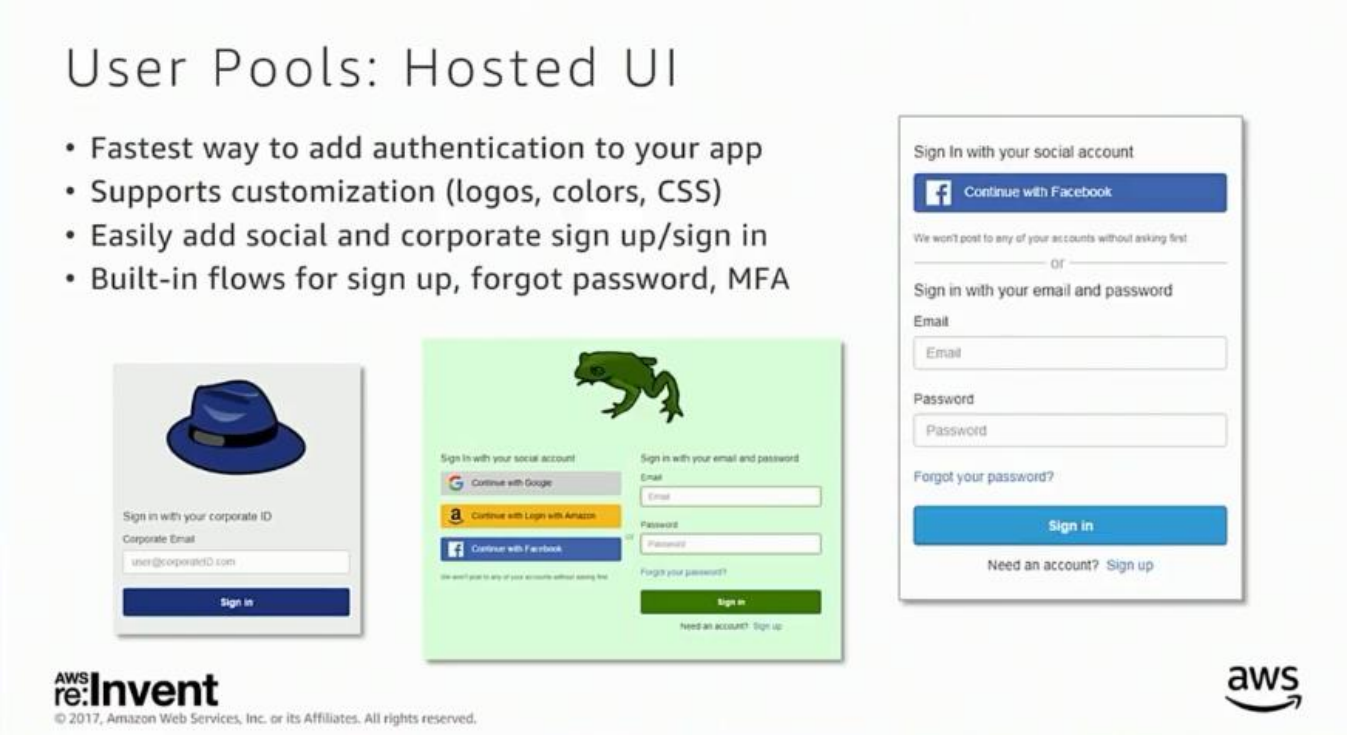
Scenario:

- Sign out a user from all apps
- Sign out a user from all devices

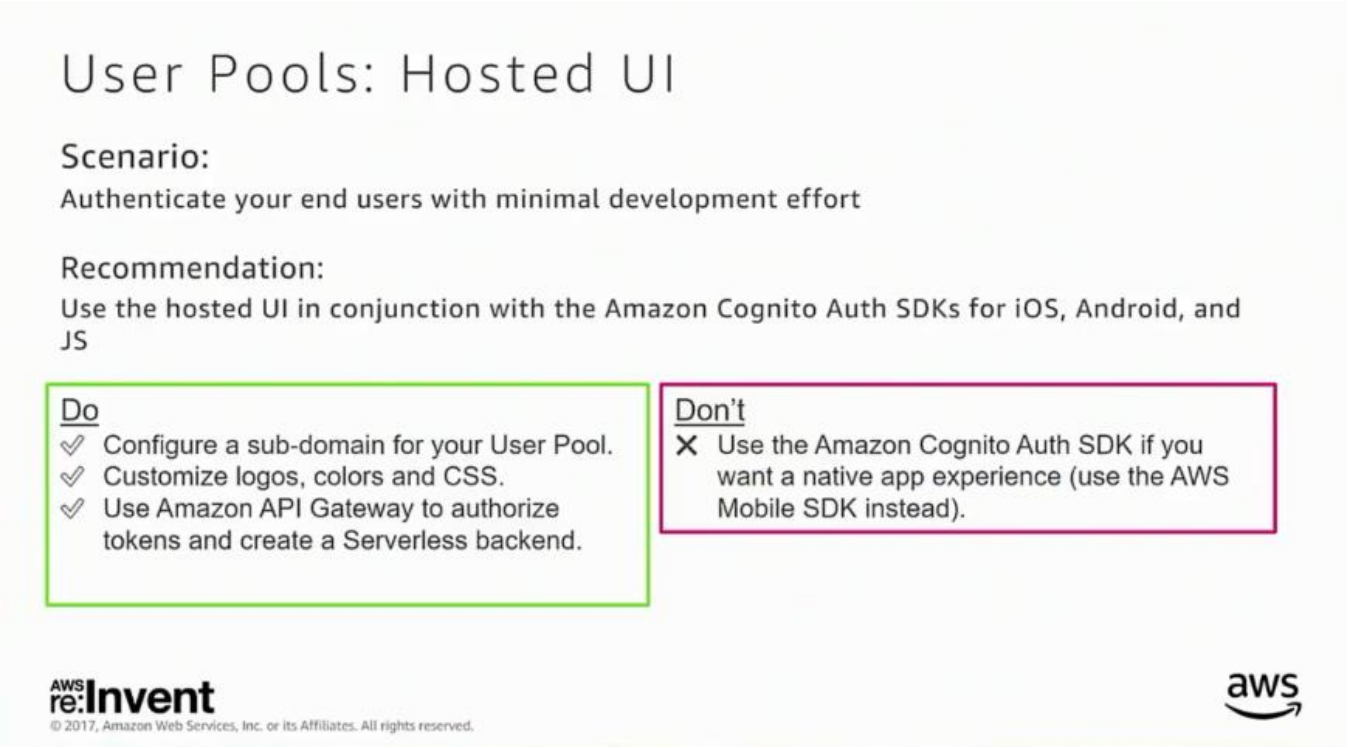
Recommendation:

- For hosted UI: use logout URI
  1. User clicks Sign Out in your app
  2. App sends a request to Amazon Cognito `https://yourdomain/logout?logout_uri=...`
  3. Amazon Cognito removes its authentication session cookie and redirects browser to your app's logout_uri
- For native SDK: call GlobalSignOut API (user) or AdminUserGlobalSignOut API (admin) to sign-out the user from all devices

Simple hosted UI, identity provider federation, and user migration



We have Hosted UIs that you can use for log in and log out in your apps, you let Cognito create the UIs for Sign up and Sign in and the related flows for user password resets for your apps. The Hosted Uis can be customized to use your logos and colors.



When you use this approach, you need to use the Cognito SDK, the user is shown the Hosted UI and after authentication you will get the tokens back from Cognito. You can use API Gateway and Lambda for Serverless backend for your apps.

User Pools: Sign in with Social IDs (1)

Scenario:

- Avoid friction of asking consumer app users to register for a new account
- Enable users to use social identities to sign in to your app
  - Facebook
  - Gmail
  - Amazon (log in with Amazon)
- Avoid developing integrations with many identity providers

Recommendation: Configure social identity providers in your User Pool

User Pools: Sign in with Social IDs (2)

How to set up User Pool for Facebook authentication:

1. Add Facebook as identity provider in your User Pool
   - Enter your app's Facebook app ID, app secret, and Facebook scopes (permissions) to be requested
2. Add User Pool domain URL in Facebook app's OAuth2 Redirect URLs
3. Map Facebook attributes to User Pool attributes
   - Standard + custom attributes are in id token, e.g. name, email
   - Map Facebook access token so your app can call Facebook APIs
4. Configure settings for your app client
   - Enable Facebook identity provider for your app client
   - Set redirect URIs, OAuth2 flows and scopes ("openid" to get id token)
5. Use Amazon Cognito Auth SDK for Android / iOS / Javascript

Sign in Enterprise Users (1)

Scenario:

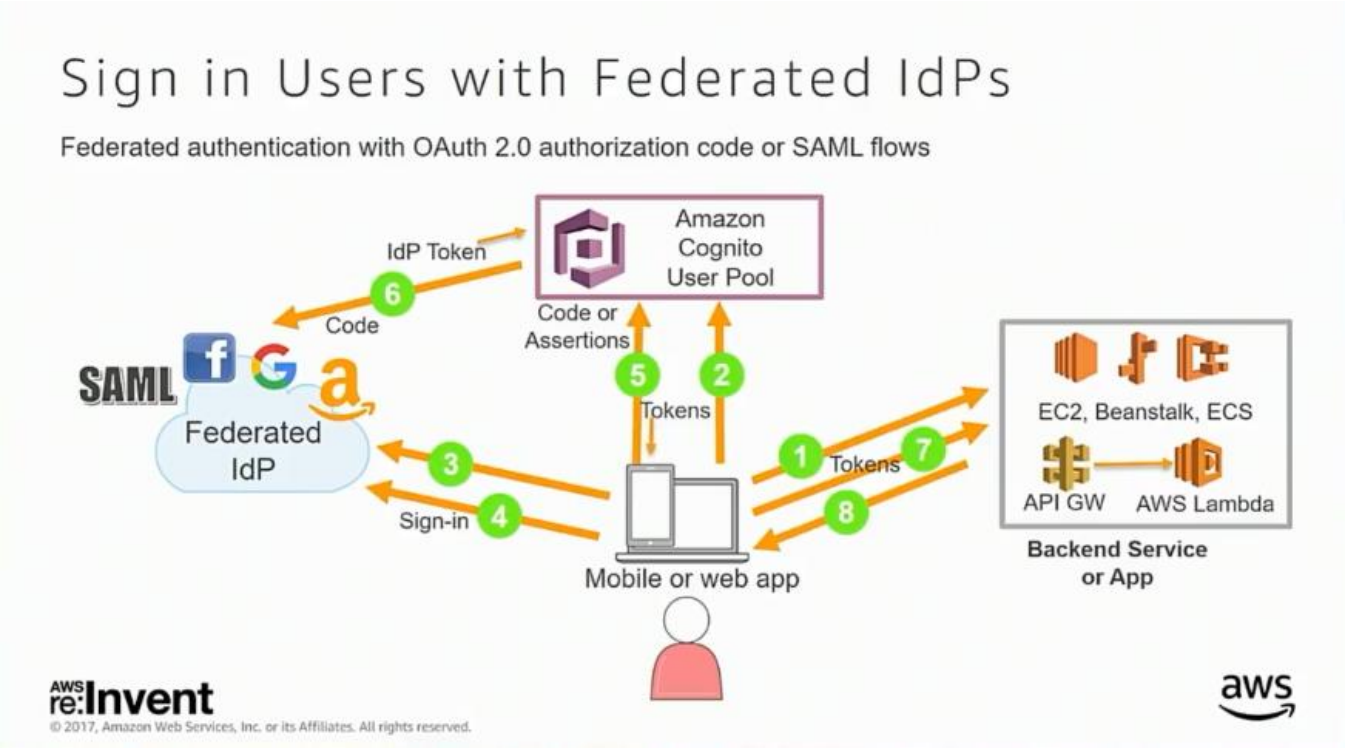
- Avoid friction of asking enterprise app users to register for a new account
- Enable users to use enterprise identities to sign in to your app
    - SAML identity providers include Microsoft ADFS, Okta, PingFederate, etc.
- Avoid developing integrations with many enterprise directories
- Avoid development effort of implementing SAML in your app
    - Constructing and parsing XML messages
    - HTTP redirect and POST bindings

Recommendation: Use User Pools SAML IdP federation.

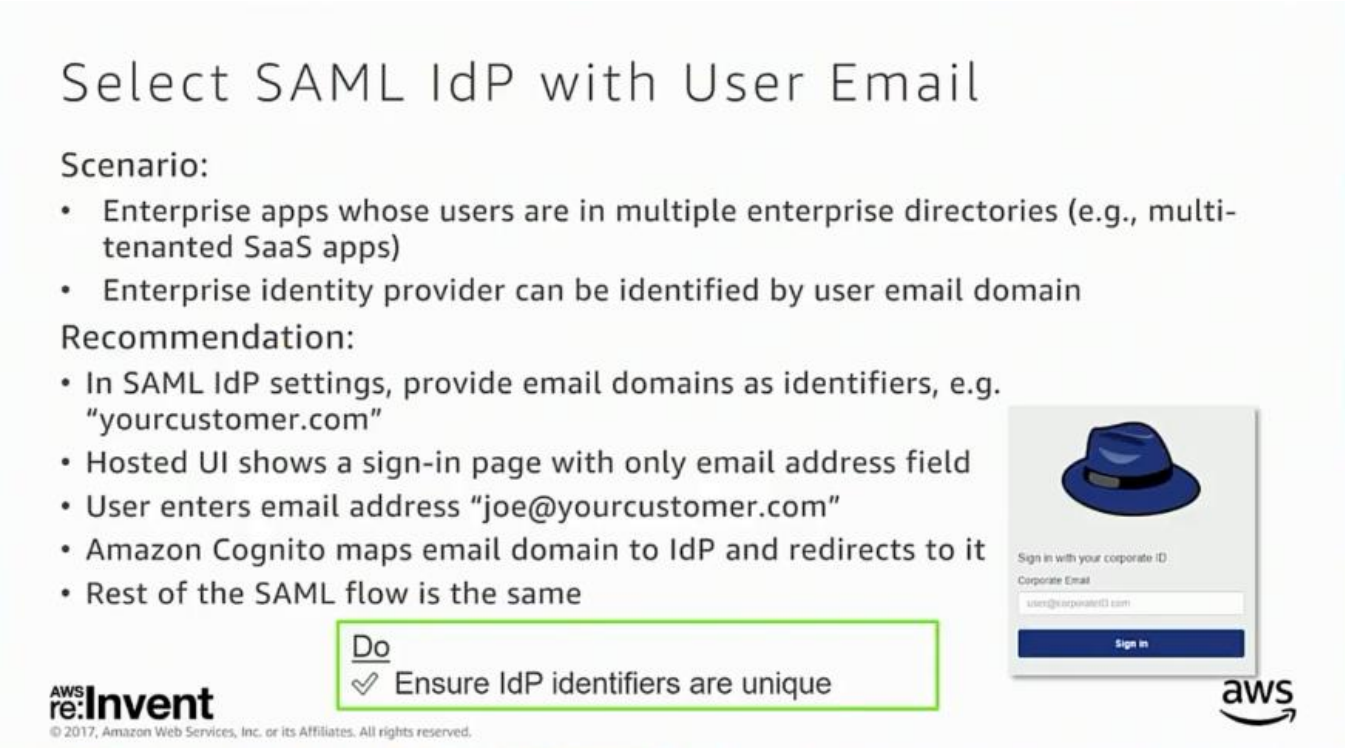
Sign in Enterprise Users (2)

How to setup User Pool for authentication with SAML enterprise identity provider:

1. Get your enterprise identity provider's SAML metadata URL or XML file
2. Add a SAML identity provider in your User Pool
    - Enter enterprise identity provider SAML metadata URL or upload XML file
3. Add User Pool URL as SAML POST binding URL in enterprise identity provider
4. Map SAML attributes to User Pool attributes
    - Standard + custom attributes are in id token, e.g. name, email
5. Configure settings for your app client
    - Enable your SAML identity provider for your app client
    - Set redirect URIs, OAuth 2.0 flows and scopes ("openid" to get id token)
6. Use Amazon Cognito Auth SDK for Android / iOS / Javascript



Above is an example of how a mobile or web app does authentication using Cognito federation



User Pools: SSO Across Apps

Scenario:

You have multiple apps but want a unified user profile and to be authenticated across all

Recommendation:

- Create a single User Pool, and one app client for each app
    - Each app has a client id, and different auth flows / attribute permissions
- After authentication, Amazon Cognito stores a cookie in user's browser
- When second app redirects browser to Amazon Cognito for sign-in, Amazon Cognito checks cookie. If valid, Amazon Cognito redirects back to your app with tokens.

Do

- ✓ Set allowed OAuth2 flows for each app
- ✓ Set attribute read/write permissions for each app

aws re:Invent

aws

Migrating Users to User Pool: Bulk

Scenario:

- Many users in existing user directory in your app
- You want to bulk migrate to Cognito User Pool

Recommendation:

- Create a CSV file with all users, one row per user, one column per user attribute
    - Must have username, verified email, or phone number attributes
- Upload CSV files using Amazon Cognito console or AWS CLI, start import job
- Track user import job using Amazon CloudWatch logs in your AWS account
- Update your mobile apps and web apps to use Amazon Cognito for sign-in
- Users will need to reset their password at first sign-in, using forgot-password flow (verification code sent to their email/phone)

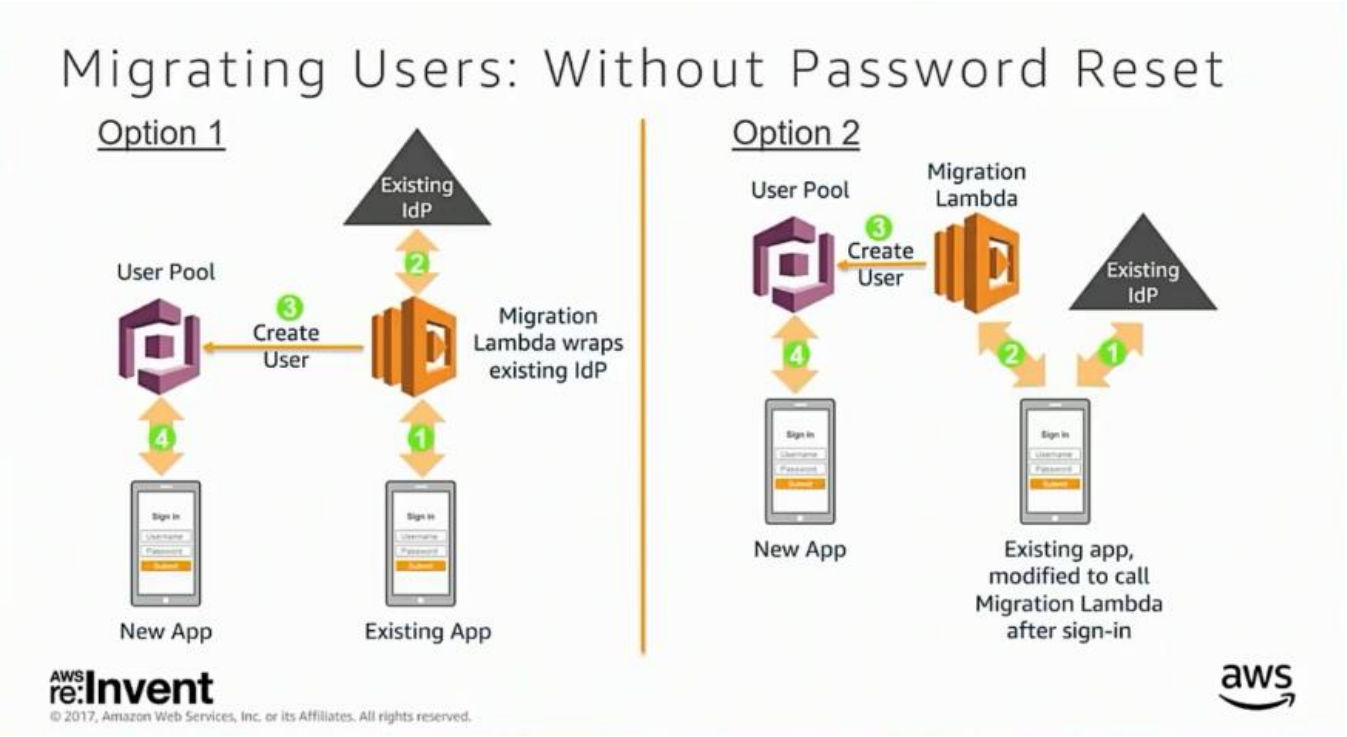
Do

- ✓ Provide IAM role for Amazon Cognito to write CloudWatch logs to your account
- ✓ Set required user attributes in CSV file

aws re:Invent

aws

The user will still need to reset their password when they log in the next time around

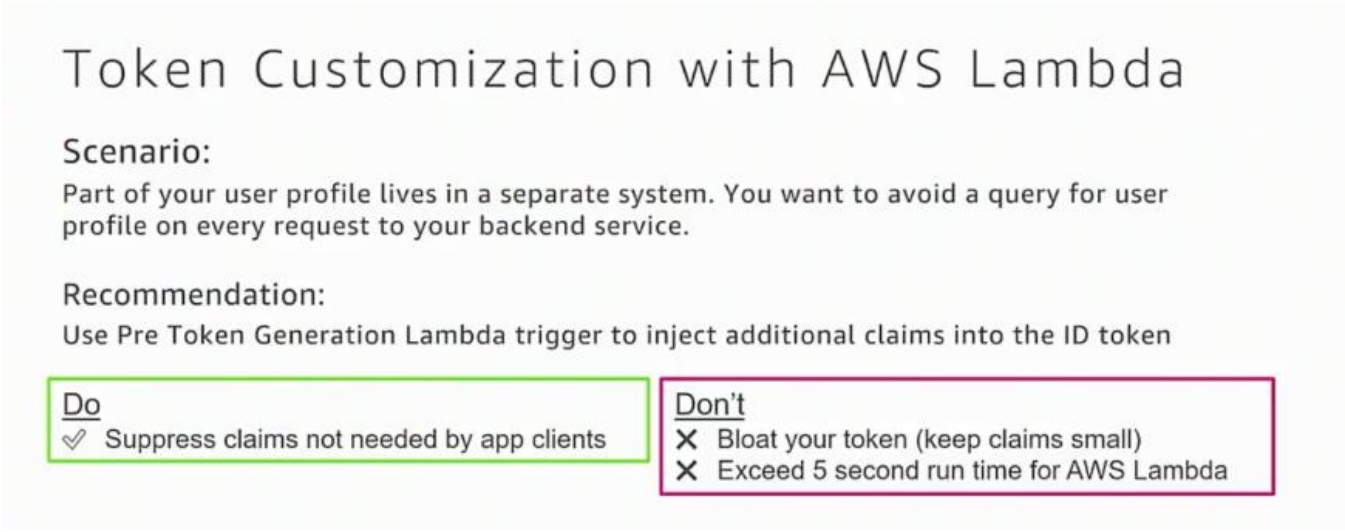


You can use one of the above options if you don't want your users to have to reset their passwords.

User Pools: Customizing with AWS Lambda

Category	Lambda Hook	Example Scenarios
Custom Authentication Flow	Define Auth Challenge	Determines the next challenge in a custom auth flow
	Create Auth Challenge	Creates a challenge in a custom auth flow
	Verify Auth Challenge Response	Determines if a response is correct in a custom auth flow
Authentication Events	Pre Authentication	Custom validation to accept or deny the sign-in request
	Post Authentication	Event logging for custom analytics
	Pre Token Generation	Augment or suppress token claims
Sign-Up	Pre Sign-up	Custom validation to accept or deny the sign-up request
	Post Confirmation	Custom welcome messages or event logging for custom analytics
Messages	Custom Message	Advanced customization and localization of messages

Amazon Cognito can be configured to use AWS Lambda at different points to call into your authentication workflows. This gives you the ability to tweak your workflows in the way you want to use Cognito, you simply create these lambdas and allow Cognito access to call them at different hook points for doing things like sending a welcome email, etc.



The accesscode can be supplied to your lambda as above

User Pools: Customizing Authentication

Scenario:

Add additional challenges into the authentication flow, e.g., secret questions, captcha

Recommendation:

- Create DefineAuthChallenge Lambda function: called in beginning and after each challenge, determines next step (challenge name, or success/failure). Can use password / MFA / custom challenge.
- Create CreateAuthChallenge Lambda function: creates challenge (e.g. captcha URL / secret question) and answer, for custom challenge.
- Create VerifyAuthChallenge: called after user inputs answer, verifies answer from user; then Amazon Cognito calls DefineAuthChallenge again.

Do
- ✓ Use built-in auth flows if you can
- ✓ Use privateChallengeParameters for answers
- ✓ Use high-level SDKs

Don't
- ✗ Exceed 5 second run time for Lambda

A User Pools SDK for Every App

Identity Pools: Getting AWS Credentials

- If you have a token for your end user, Identity Pools can exchange it for temporary AWS Credentials
- You use IAM roles to define what AWS resources your user can access directly
- There are two roles by default: unauthenticated and authenticated
- You can use role mappings to map a claim in a token to a specific role
- The returned AWS Credentials contain the chosen role

Identity Pools: Getting AWS Credentials

Scenario:

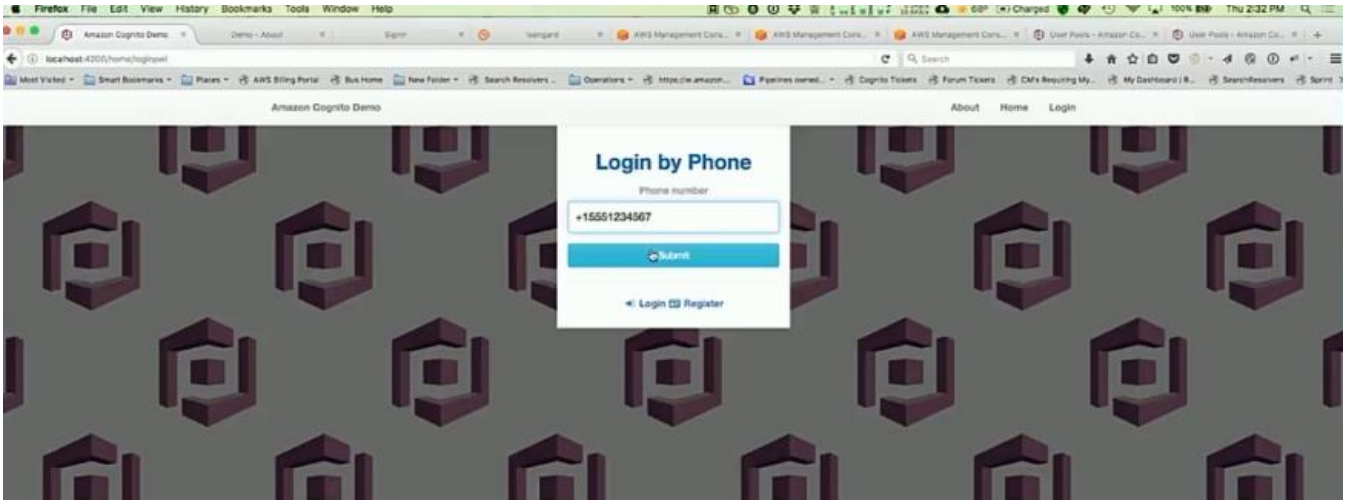
- Give User Pool users in admin group access to entire Amazon S3 bucket

Recommendation:

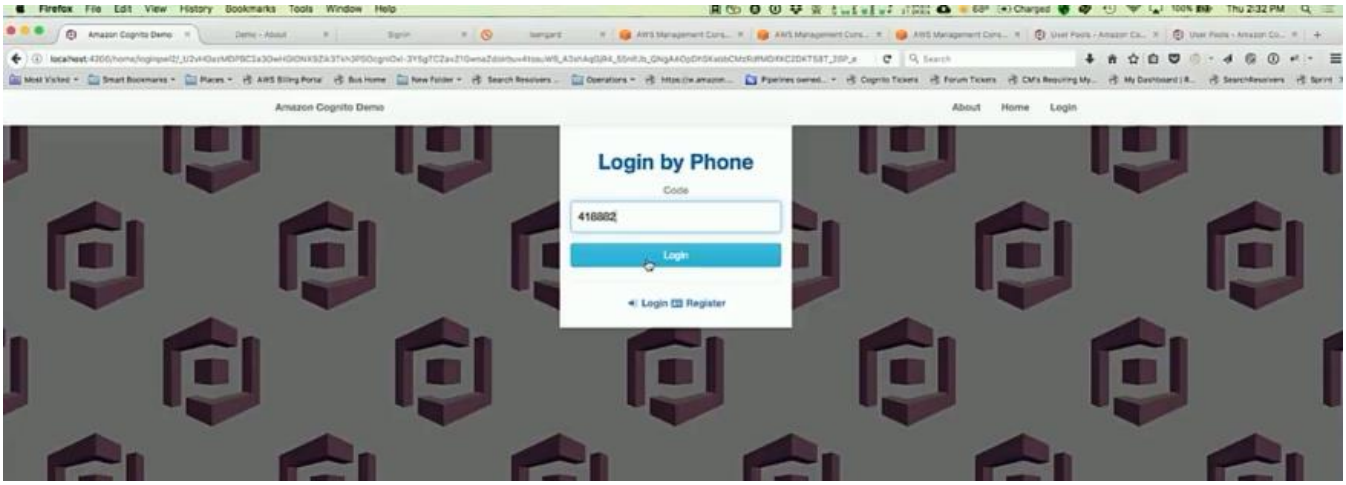
- Create an IAM Role that grants access to the Amazon S3 bucket
- Configure your admin group in User Pools with your admin role
- Configure Identity Pool's Role Mapping to use role from the User Pool ID token

Do

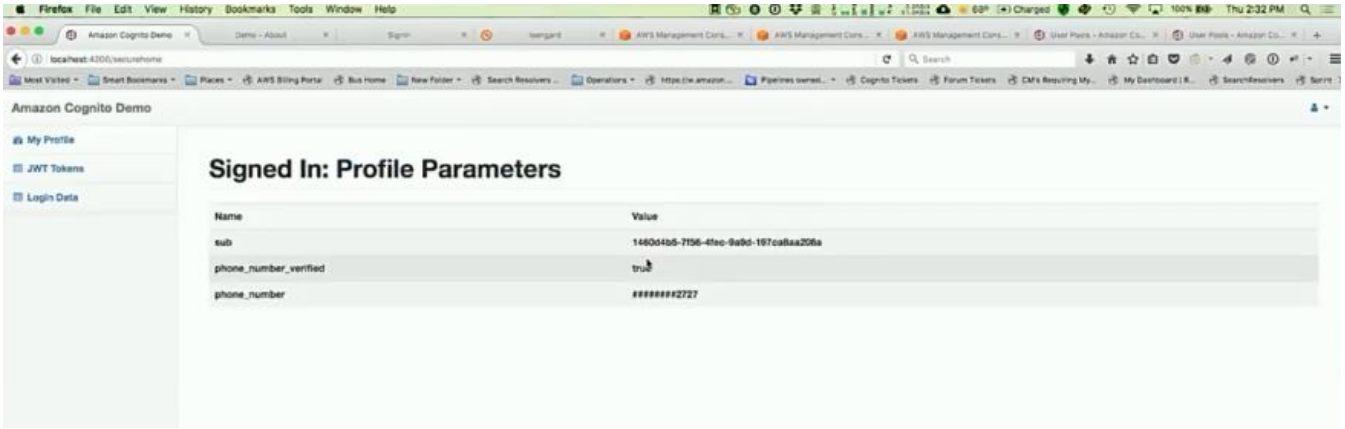
✓ Limit permissions for unauthenticated users



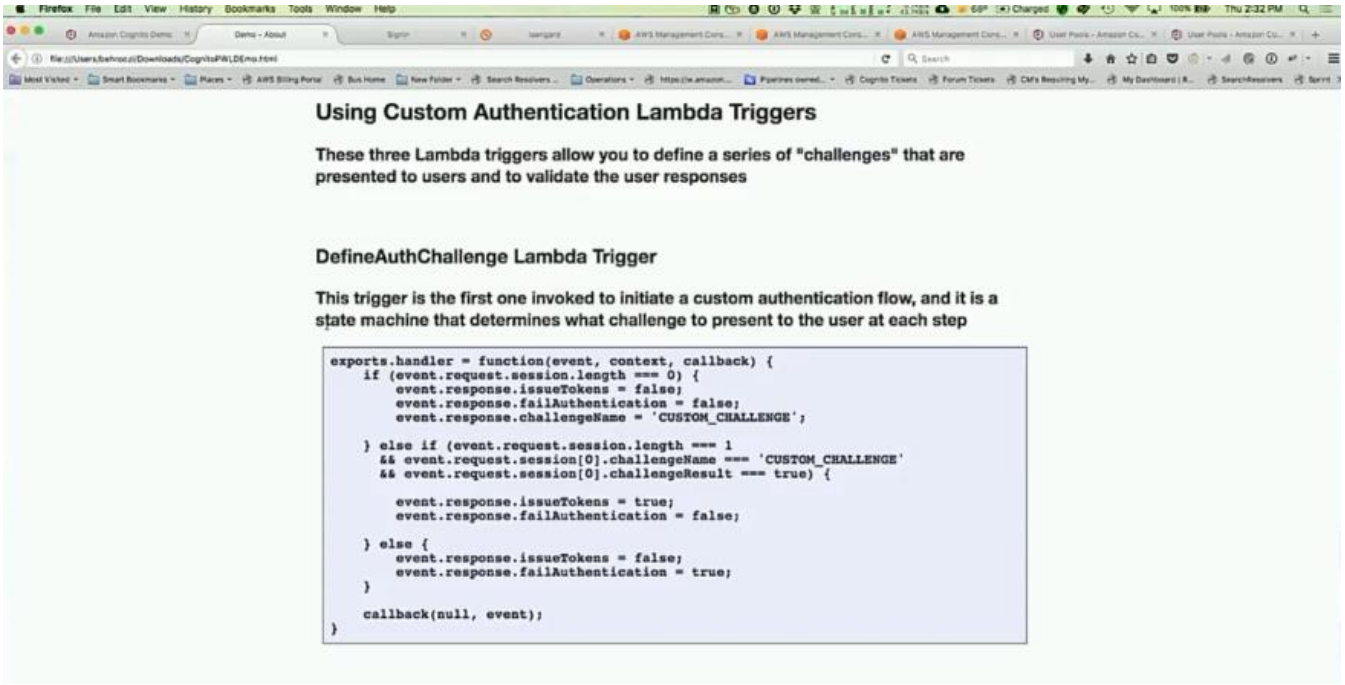
We rare going to be demoing this custom authentication flow, imagine a scenario where you want to send a code to the user's phone. When the user clicks the Submit button, it will invoke a lambda function that will send the user a text message on their phone.



The user can then enter the code sent in the message to log into the app



The user is now signed in and we get the details on the landing page



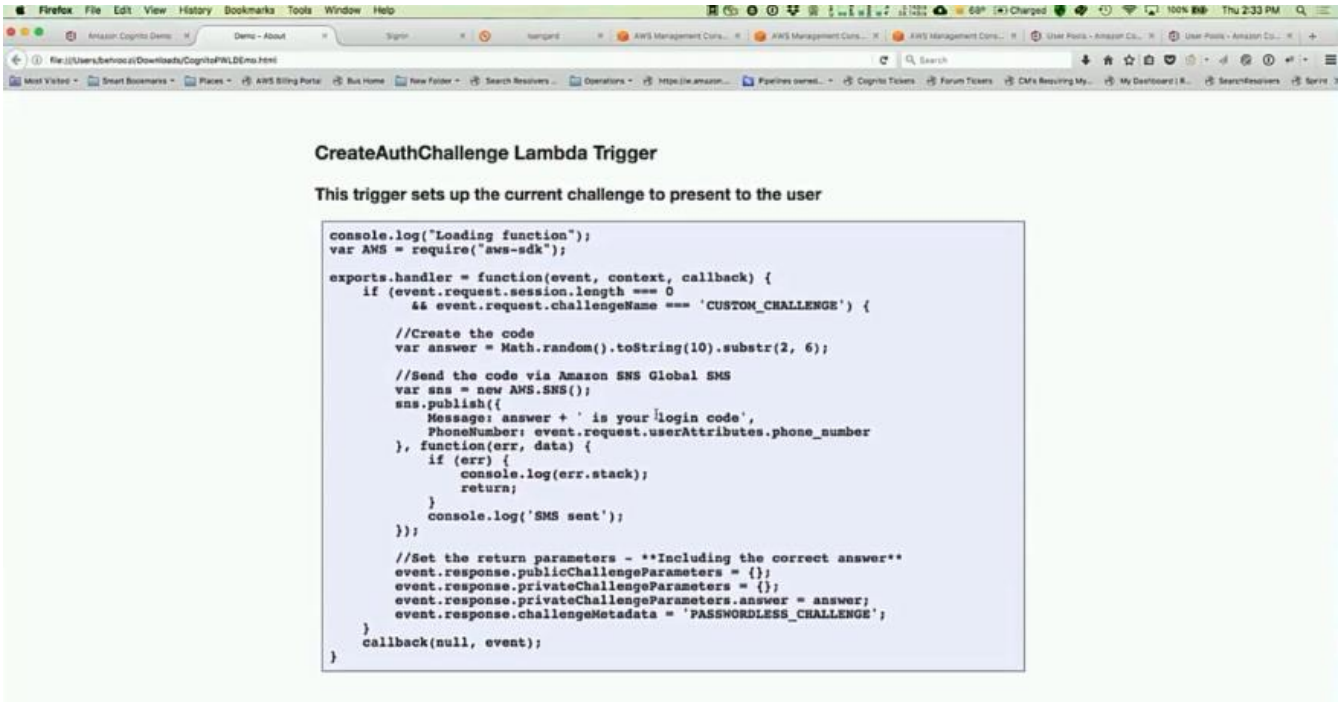
Using Custom Authentication Lambda Triggers

These three Lambda triggers allow you to define a series of "challenges" that are presented to users and to validate the user responses

DefineAuthChallenge Lambda Trigger

This trigger is the first one invoked to initiate a custom authentication flow, and it is a state machine that determines what challenge to present to the user at each step

```
exports.handler = function(event, context, callback) {
    if (event.request.session.length === 0) {
        event.response.issueTokens = false;
        event.response.failAuthentication = false;
        event.response.challengeName = 'CUSTOM_CHALLENGE';

    } else if (event.request.session.length === 1
      && event.request.session[0].challengeName === 'CUSTOM_CHALLENGE'
      && event.request.session[0].challengeResult === true) {

        event.response.issueTokens = true;
        event.response.failAuthentication = false;

    } else {
        event.response.issueTokens = false;
        event.response.failAuthentication = true;
    }

    callback(null, event);
}
```

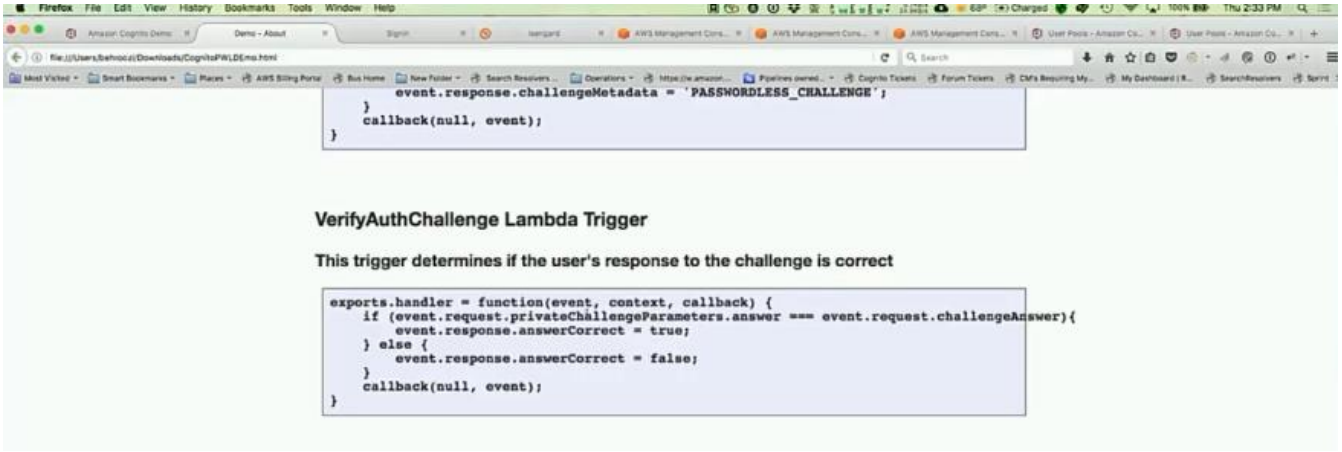


CreateAuthChallenge Lambda Trigger

This trigger sets up the current challenge to present to the user

```
console.log("Loading function");
var AWS = require("aws-sdk");

exports.handler = function(event, context, callback) {
    if (event.request.session.length === 0
          && event.request.challengeName === 'CUSTOM_CHALLENGE') {

        //Create the code
        var answer = Math.random().toString(10).substr(2, 6);

        //Send the code via Amazon SNS Global SMS
        var sns = new AWS.SNS();
        sns.publish({
            Message: answer + ' is your login code',
            PhoneNumber: event.request.userAttributes.phone_number
        }, function(err, data) {
            if (err) {
                console.log(err.stack);
                return;
            }
            console.log('SMS sent');
        });

        //Set the return parameters - **Including the correct answer**
        event.response.publicChallengeParameters = {};
        event.response.privateChallengeParameters = {};
        event.response.privateChallengeParameters.answer = answer;
        event.response.challengeMetadata = 'PASSWORDLESS_CHALLENGE';
    }
    callback(null, event);
}
```

This is the sample code that actually implements sending the text message via SNS

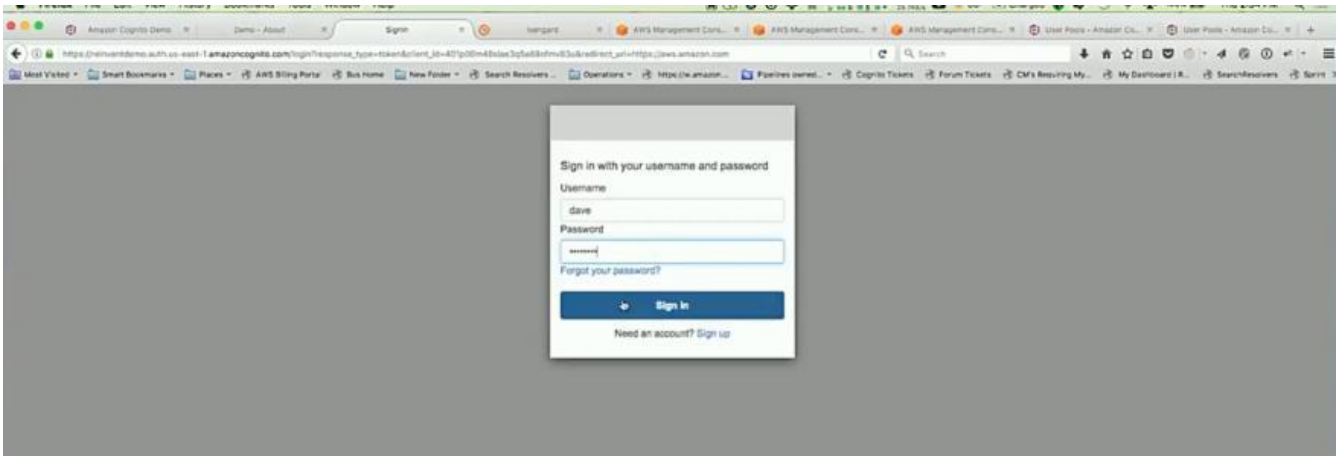


```
        event.response.challengeMetadata = 'PASSWORDLESS_CHALLENGE';
    }
    callback(null, event);
}
```

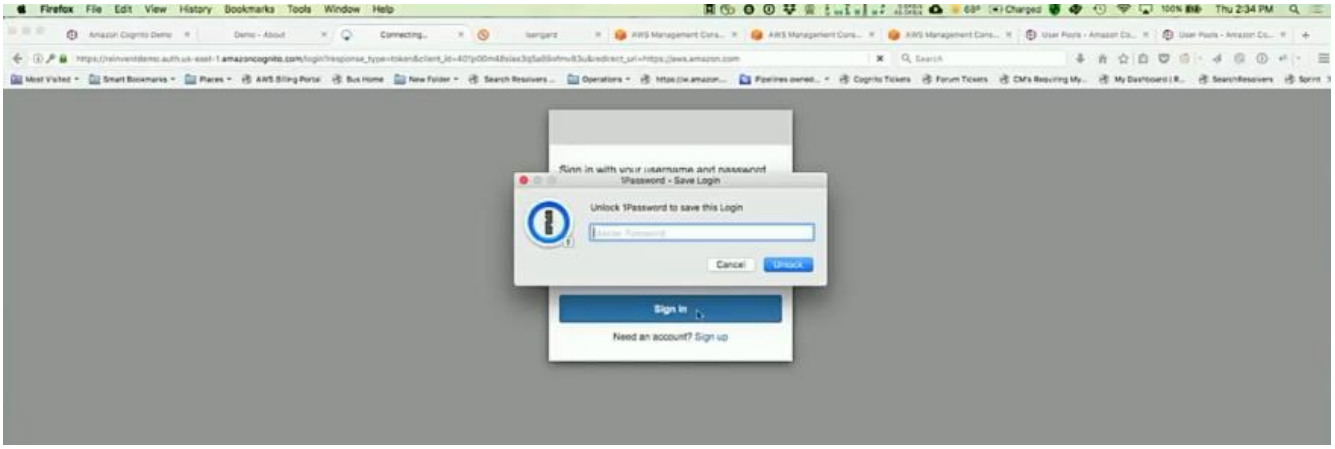
VerifyAuthChallenge Lambda Trigger

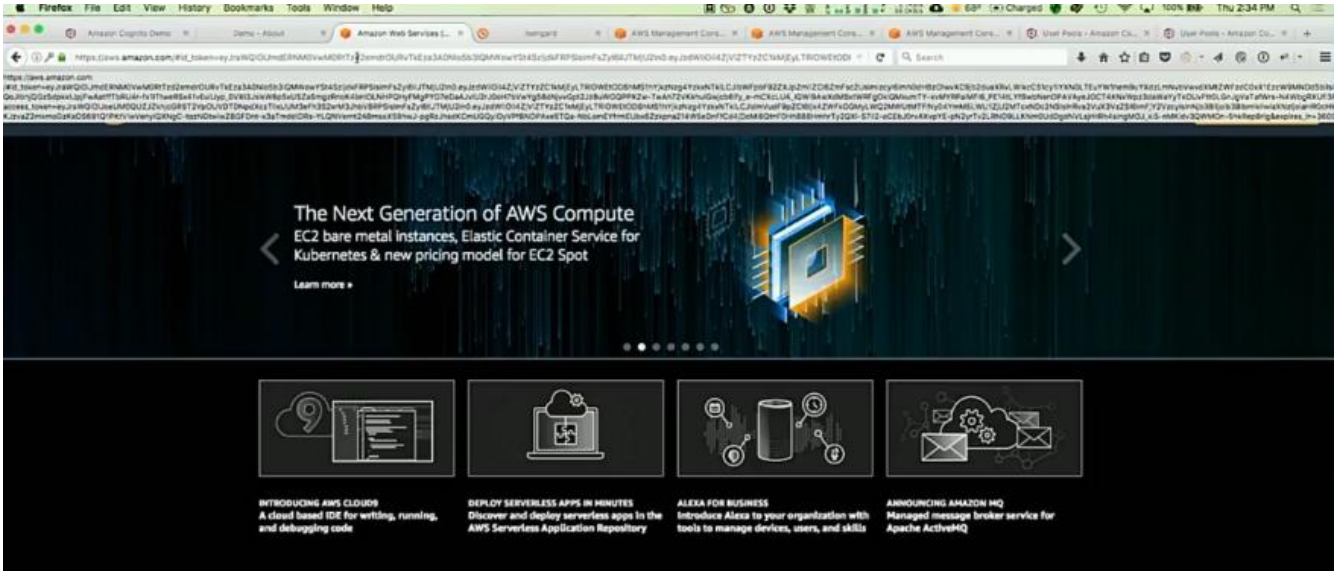
This trigger determines if the user's response to the challenge is correct

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

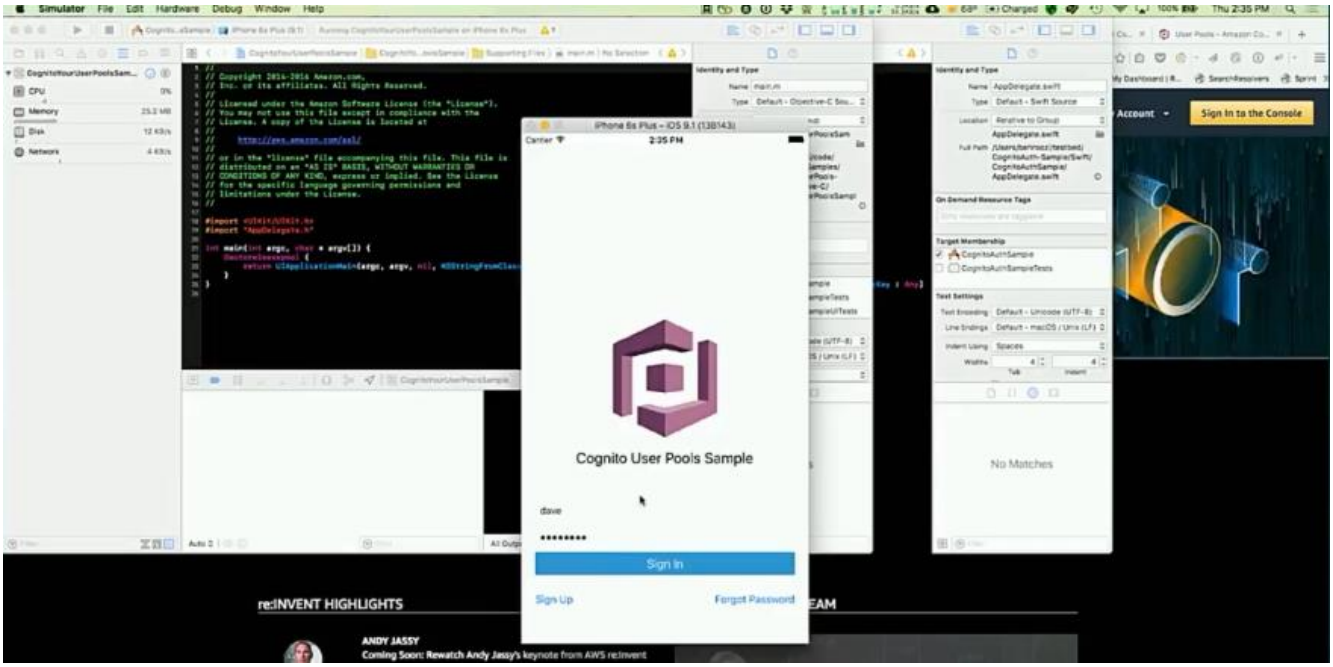


This demo is going to show adaptive authentication, we are going to authenticate with a couple of different apps and websites using our Hosted UI approach

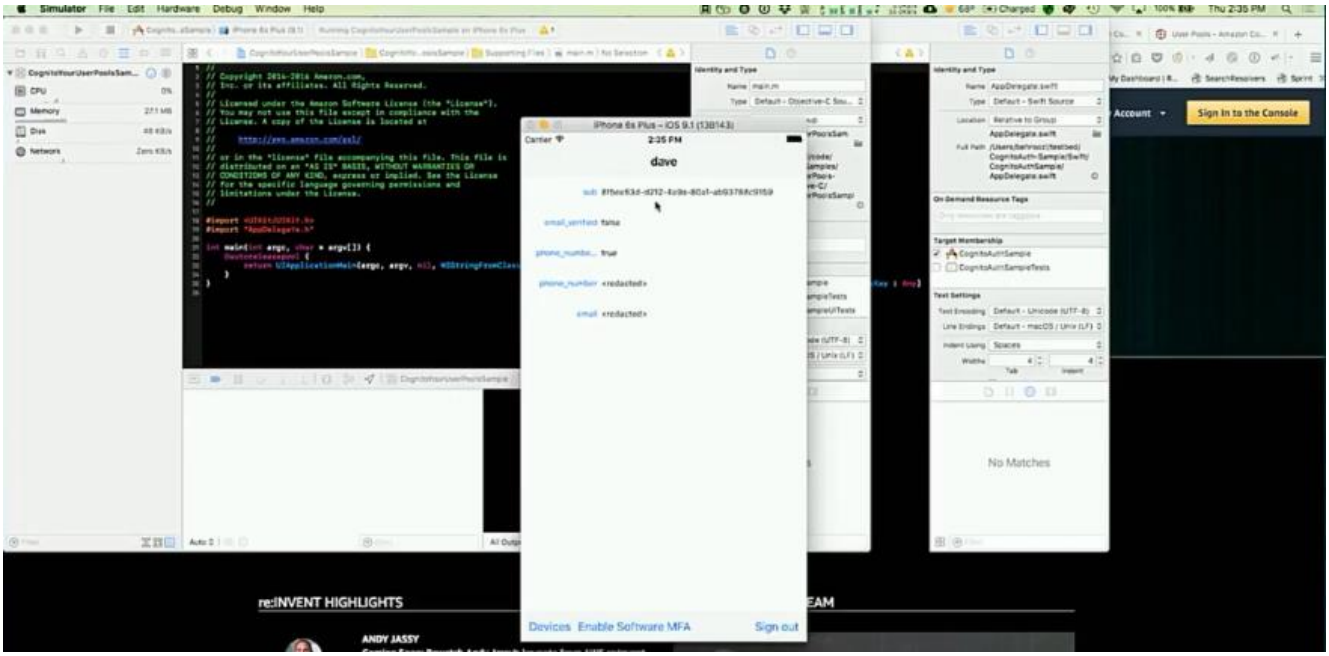




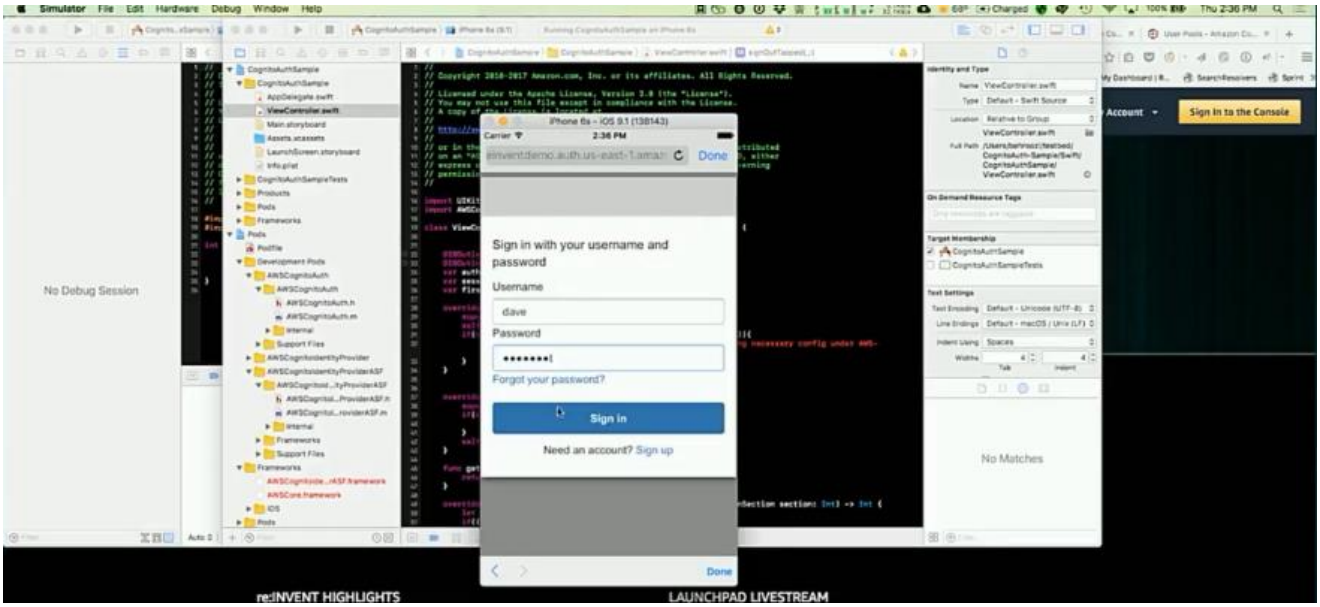
It then redirects you to the website you configured along with the authentication token from Cognito for authenticating

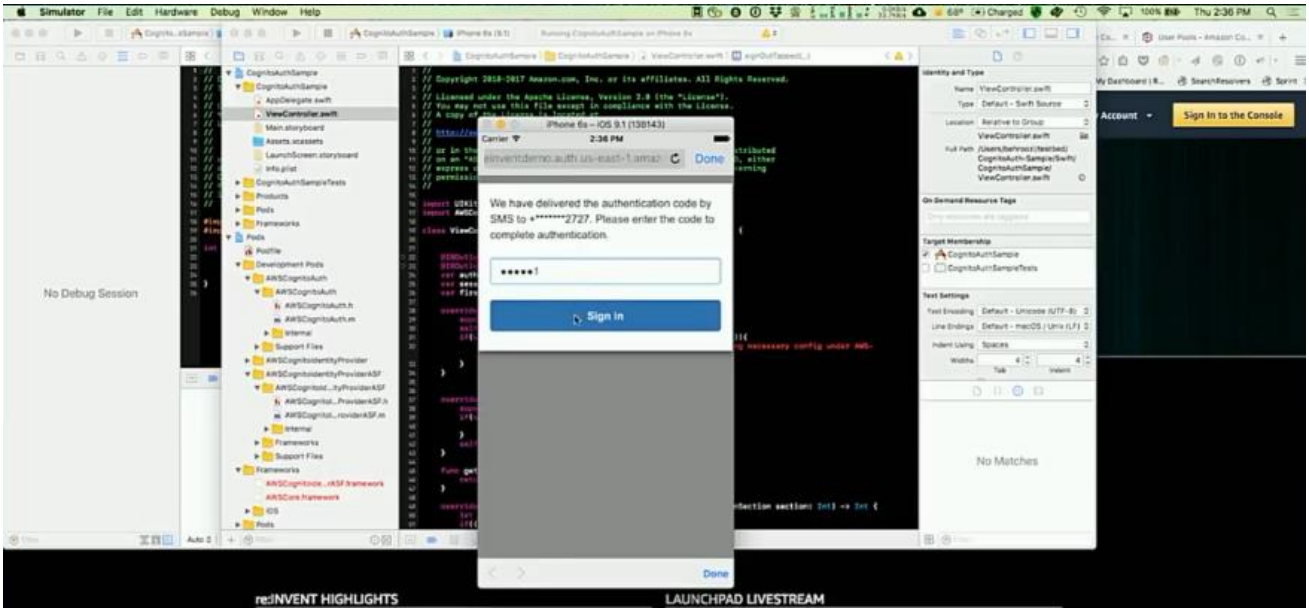


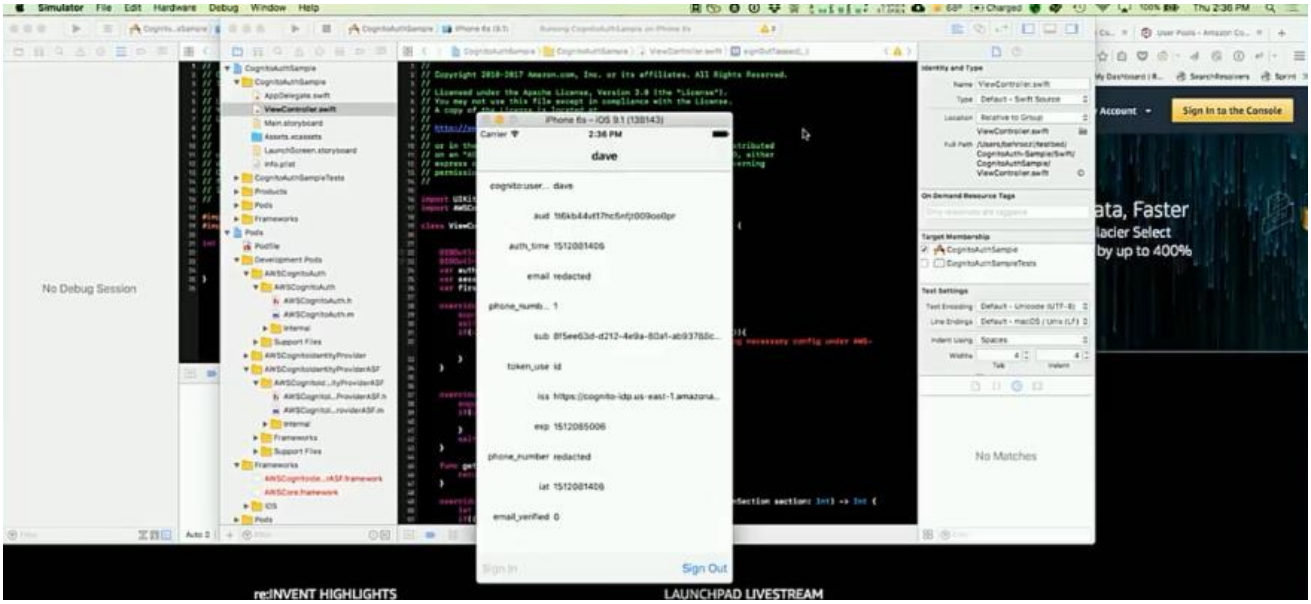
This is using our native SDKs for IOS



We are again signed in successfully







We are again signed in

Amazon Cognito Summary

- Secure authentication and authorization
- Simple hosted UI, identity provider federation, and user migration
- Flexible app integration with custom authentication flows and UIs

Additional Resources

See aws.amazon.com/cognito/dev-resources/ for links to:

- Getting started guides
- Documentation, SDKs, and sample apps
- Videos
- Presentation slides
- Blog posts
- Developer forums