Practical: AWS Cognito (Javascript)

Table of Contents

This practical introduces you to AWS Cognito, which is a service for managing user identities and authentication. Cognito is quite complex, so we'll only introduce you to a small subset of its functionality which will allow you to sign up users, confirm their email address, and authenticate them.

Prerequisites and references

- **AWS Cognito docs** ⬈ **(https://docs.aws.amazon.com/cognito/)**
- **AWS SDK Congito API docs** ⬈ **(https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/cognito-identity-provider/)**
- **AWS SDK Cognito code examples** ⬈ **(https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javascriptv3/example_code/cognito-identity-provider)**
- **AWS JWT verifier** ⬈ **(https://github.com/awslabs/aws-jwt-verify)**

AWS Cognito fundamentals

The core object in Cognito is *User pools*. A user pool is basically a database holding on to user identity information such as username, password, email address, and so on. To use Cognito you start by creating a user pool. The user pool configuration controls various aspects of identity and authentication such as what identity information is kept, whether users need to be confirmed with an email or SMS, multifactor authentication, and so on.

Cognito's model allows for several *App clients* to interface with a user pool. For example, you might have web and mobile clients that both need to authenticate their users. A user pool includes a list of app clients and the configuration associated with each app client, such as authentication methods.

There is a lot of additional functionality in AWS that we will leave for you to explore on your own. Here are some examples:

- Multi-factor authentication
- Integration with federated identity providers (eg. Google, Facebook, etc.)
- User groups
- Integration with AWS IAM roles for authorisation.

Creating a Cognito user pool

We'll start the practical by setting up a user pool. We'll keep things simple.

- Find Cognito in the AWS console
- *Create user pool*.
- *Define your application:*
  - For *Application Type* choose *Traditional web application*
  - Add a name such as *n1234567-prac*
- *Configure options*
  - Select *Username* for sign-in options
  - Under *Required attributes for sign-up* select *email*
- *Add a return URL*:
  - Leave this as is for now
- *Create user directory*

You should now see a page that offers example code for setting up authentication with a single-page web app. You can come back to this if you have a web client for your application. Now we'll see some more configuration options for Cognito.

- In the sidebar, click on *User pools*
- Find your user pool by the name that you gave it and click on the name to bring up the details page.
- Take note of the *User pool ID* in the *Overview* page. You will need this later.
- Click on *App clients* in the sidebar
  - You should see one app client in the list. Click on its name.
  - Here you can find example code for the server component of a single-page application
  - Take note of the *Client ID* and *Client secret*, which are used to log in. You will need these later.
  - Click on *Edit* in the top right part of the *App client information* section
    - Under *Authentication flows* tick *Sign in with username and password: ALLOW_USER_PASSWORD_AUTH* so that we can use password authentication.
    - Click *Save changes*

Finally, we need to set up the email system for confirming emails. The default email configuration is limited to 50 emails per day across our AWS account, which will quickly be used up. If you get errors about email limits, check that you have set the email configuration to use SES.

- In the sidebar, click *Authentication*

- Choose *Send Email with Amazon SES*
- Under *FROM email address* choose *cab432.com*, which should be the only option
- Under *FROM sender name* enter `CAB432 logins <logins@cab432.com>`. You can change the name and mailbox to something different if you like, but the email address must end with `@cab432.com`.

There are multiple configuration options and management operations that are supported in Cognito. While on the details page for your user pool, have a look around at the items in the sidebar, especially *Users*.

Setting up the client

There are a number of different ways of authenticating with Cognito. We'll demonstrate one method which can be used either from client-side javascript or server-side.

API calls

There are many API calls that you can make (see **AWS SDK Congito API docs** ⬈ **(https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/cognito-identity-provider/)** for more info.) We'll explore three in this practical.

- `SignUpCommand` : Sign up a new user
- `ConfirmSignupCommand` : Confirm a new user using a code from a confirmation email
- `InitiateAuthCommand` : Authenticate an existing user

Signing up

Start by downloading the sample code: **cognitodemo.zip (https://canvas.qut.edu.au/courses/20367/files/6903230/download)**. You can run this on your local computer if you have Node installed, or on an EC2 instance. If you are running it on an EC2 instance then unzip and use `scp` or similar to upload it to your EC2 instance. For the remainder we assume that you have a terminal open to your local computer, or an SSH connection to EC2 as appropriate, and that the working directory is the Cognito demo directory.

- `npm install` to install the dependencies
- Edit `signUp.js` and change the following:
  - `clientId` set to the client ID from the *App Client* details page
  - `clientSecret` set to the client secret from the *App Client* details page
  - `username` set to a username of your choice. For example `matt`
  - `password` set to a password of your choice. It will need to meet the default Cognito password requirements:
    - 8 character(s) minimum
    - Contains at least 1 number
    - Contains at least 1 uppercase letter
    - Contains at least 1 lowercase letter
    - Contains at least 1 symbol

- `email` set to an email address that you can access to retrieve the confirmation code
- run `node signUp.js`

The script sends the username, email address and password to the Cognito service which will create a user in the user pool. It will then send out a confirmation email containing a code.

At this point you can return to the AWS console. Find your user pool from the list and scroll down to *Users*. You should now see a user there. Note that in the *Email verified* column it will say "No".

Confirming signup

This step confirms that the user controls the email account that they specified on signup.

- Access the email account that you specified in the previous section. Retrieve the code.
- Edit `confirm.js` and change the following:
  - `clientId` to match that in `signUp.js`
  - `clientSecret` to match that in `signUp.js`
  - `username` to match that in `signUp.js`
  - `confirmationCode` set to the code your retrieved from the confirmation email
- run `node confirm.js`

This script will confirm the user's email address using the code. If you go to the AWS Console to the *Users* list, the *Email Verified* column will now say "Yes". You may need to refresh the list to see this.

Authenticating a user

After a user has had their email confirmed they can log in with their username and password.

- Edit `authenticate.js` and change the following:
  - `userPoolId` : set to the user pool ID you obtained from the AWS console previosuly.
  - `clientId` : set to the client ID you obtained from the AWS console previously.
  - `username` : set to the username as in `signUp.js`
  - `password` : set to the password as in `signUp.js`
- run `node authenticate.js`

This will obtain two JWTs which authenticate the user. It then verifies the JWTs and prints out the data that they contain.

There are two JWTs returned:

- `AccessToken` : this token can be used to authenticate to AWS services if the user is associated with an IAM role. We will not be using it as we don't have permissions to create such IAM roles. You can ignore this token.
- `IdToken` : this token contains user information such as their username and email address. If other identity information is stored in the user pool then it will be available in the token.

Integrating into an app

Here we sketch how you can use Cognito with your application, which will be part of assessment 2.

- Create endpoints for signing up, confirming, and authentication. These should mimic the functionality of `signUp.js`, `confirm.js` and `authenticate.js`
- For endpoints which require authentication, your API server should check that requests contain an appropriate header containing a valid JWT. Your server can also retrieve the username from the JWT to provide information relevant to that user (for example, the `/videos` endpoint retrieves the list of videos that the user owns)

If you have already used JWTs in your application, then you will likely have much of the structure already done. You will need to modify your code to use AWS calls to obtain and verify the JWTs.

Delete unused resources

When you are finished, please delete the user pool that you created. Alternatively, you can keep it around for use with your application (or create a new one.)