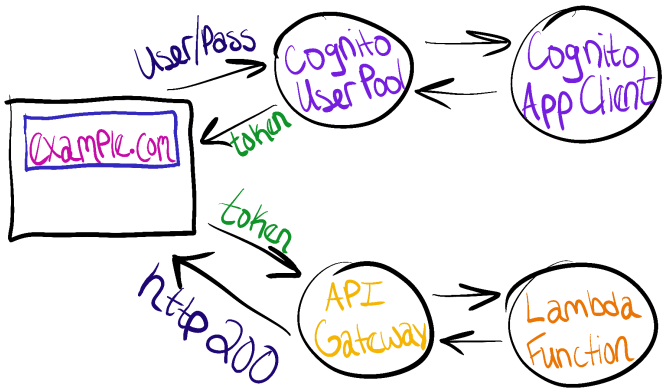
How to add Cognito to your AWS SAM app

The AWS Serverless Application Model (SAM) is a great way to start building APIs and other applications, but API endpoints are open by default. Amazon Cognito is a solution to add user sign up and sign in to a project. By the end of this post you will have created an API endpoint that requires authentication, registered a user, and called the endpoint.



Contents

The Ingredients

The following resources can be added to any AWS SAM application. An example application can be found on GitHub.

User Pool

Resource Documentation

The Amazon Cognito user pool is a collection of users. There are options for users to authenticate through social platforms or SAML, but for this example we'll have AWS store the usernames and passwords itself.

```
UserPool:
  Type: AWS::Cognito::UserPool
  Properties:
    UserPoolName: MyUserPool
    UsernameAttributes:
      - email
    Policies:
      PasswordPolicy:
        MinimumLength: 8
    Schema:
      - AttributeDataType: String
        Name: email
        Required: false
```

User Pool App Client

Resource Documentation

> *An app is an entity within a user pool that has permission to call unauthenticated API operations. Unauthenticated API operations are those that do not have an authenticated user. Examples include operations to register, sign in, and handle forgotten passwords.*

If there's one thing to understand after this blog post, it's the app client and authentication flows. Amazon Cognito supports several flows. If none are specified using the property `ExplicitAuthFlows`, then `ALLOW_CUSTOM_AUTH`, `ALLOW_USER_SRP_AUTH`, and `ALLOW_REFRESH_TOKEN_AUTH` are used. `ALLOW_REFRESH_TOKEN_AUTH` is always required.

In this example, `ALLOW_USER_PASSWORD_AUTH` is used. The client sends the username and plaintext password to Cognito. A more secure flow is recommended for production use. For other options see User pool authentication flow.

```
  UserPoolClient:
    Type: AWS::Cognito::UserPoolClient
    Properties:
      UserPoolId: !Ref UserPool
      GenerateSecret: false
      ExplicitAuthFlows:
        - ALLOW_USER_PASSWORD_AUTH
        - ALLOW_REFRESH_TOKEN_AUTH
```

User Pool User

Resource Documentation

In this section we create an initial user rather than signing up through the application (which may not totally exist). The email will be provided in the Variables section. That email will receive a temporary password.

```
  UserPoolUser:
    Type: AWS::Cognito::UserPoolUser
    Properties:
      DesiredDeliveryMediums:
        - EMAIL
      Username: !Ref CognitoUserEmail
      UserPoolId: !Ref UserPool
```

Serverless API

Resource Documentation

AWS SAM creates an API Gateway resource implicitly. We can specify it ourselves to have more control. The `Auth` section sets the User Pool as an authorizer which can then be added to specific functions.

```
  AppApi:
    DependsOn: UserPool
    Type: AWS::Serverless::Api
    Properties:
      Name: MyAppName
      StageName: !Ref APIStageName
      Cors: "'*'"
      Auth:
        Authorizers:
            CognitoAuthorizer:
              UserPoolArn: !GetAtt "UserPool.Arn"
```

Function

Resource Documentation

This block assume a `AWS::Serverless::Function` resource already exists. Appending the `RestApiId` and `Auth` fields will enforce authentication on the endpoint. If the function should stay open and not require authentication, only add `RestApiId`.

```
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Events:
        MyEvent:
          Properties:
            Path: /myfunction
            Method: GET
            RestApiId: !Ref AppApi
            Auth:
              Authorizer: CognitoAuthorizer
```

Variables

CloudFormation parameters can be used to pass in environment variables. `Parameters` should be a top level field along with `Globals` and `Resources`. `APIStageName` is hardcoded as `api` in this example but could be set to a version number or specify a dev/prod environment.

```
  Parameters:
    CognitoUserEmail:
      Description: Email address of the created user
      Type: String
    APIStageName:
      Default: api
      Description: StageName of API Gateway deployment
      Type: String
```

Try it Out

The API endpoints and HTTP responses in this example are from sam-cognito-example. The AWS CLI commands are the same for any project as long the Cloudformation resources above were used.

```
export COGNITO_USER_EMAIL='me@example.com'

sam build && sam deploy --parameter-overrides CognitoUserEmail=$COGNI
```

Make note of all of the outputs.

Unauthenticated Requests

After deployment, try a request to both endpoints. If everything went as expected, there will be two different responses.

```
GET /hello/

{"message": "hello world"}

GET /hellowithauth/

{"message":"Unauthorized"}
```

First Time Sign In

Check the inbox of `$COGNITO_USER_EMAIL` for a temporary password. This command will sign in for the first time.

```
aws cognito-idp initiate-auth --auth-flow USER_PASSWORD_AUTH --auth-p
```

Use the output in the next command. This command will set a new password and provide the final token.

```
aws cognito-idp admin-respond-to-auth-challenge --user-pool-id <USER-
```

Several tokens are provided. The ID Token is the one that will be sent with requests.

Authenticated Requests

The provided token can be sent in the `Authorization` header of each request.

```
curl -H "Authorization: Bearer <ID-TOKEN>" https://<API-ID>.execute-a
```

Recap

The CloudFormation included in this post creates the resources necessary to put API endpoints behind authentication. The resources are:

- `AWS::Cognito::UserPool`
- `AWS::Cognito::UserPoolClient`
- `AWS::Cognito::UserPoolUser`
- `AWS::Serverless::Api`

New users receive a temporary password. The user must authenticate and change their password. They then receive a token which can be sent in the `Authorization` header with all requests.

Resources

Example on GitHub

Cognito Cloudformation Reference

AWS CLI cognito-idp Reference

Written on April 5, 2022

Helpful Not Helpful