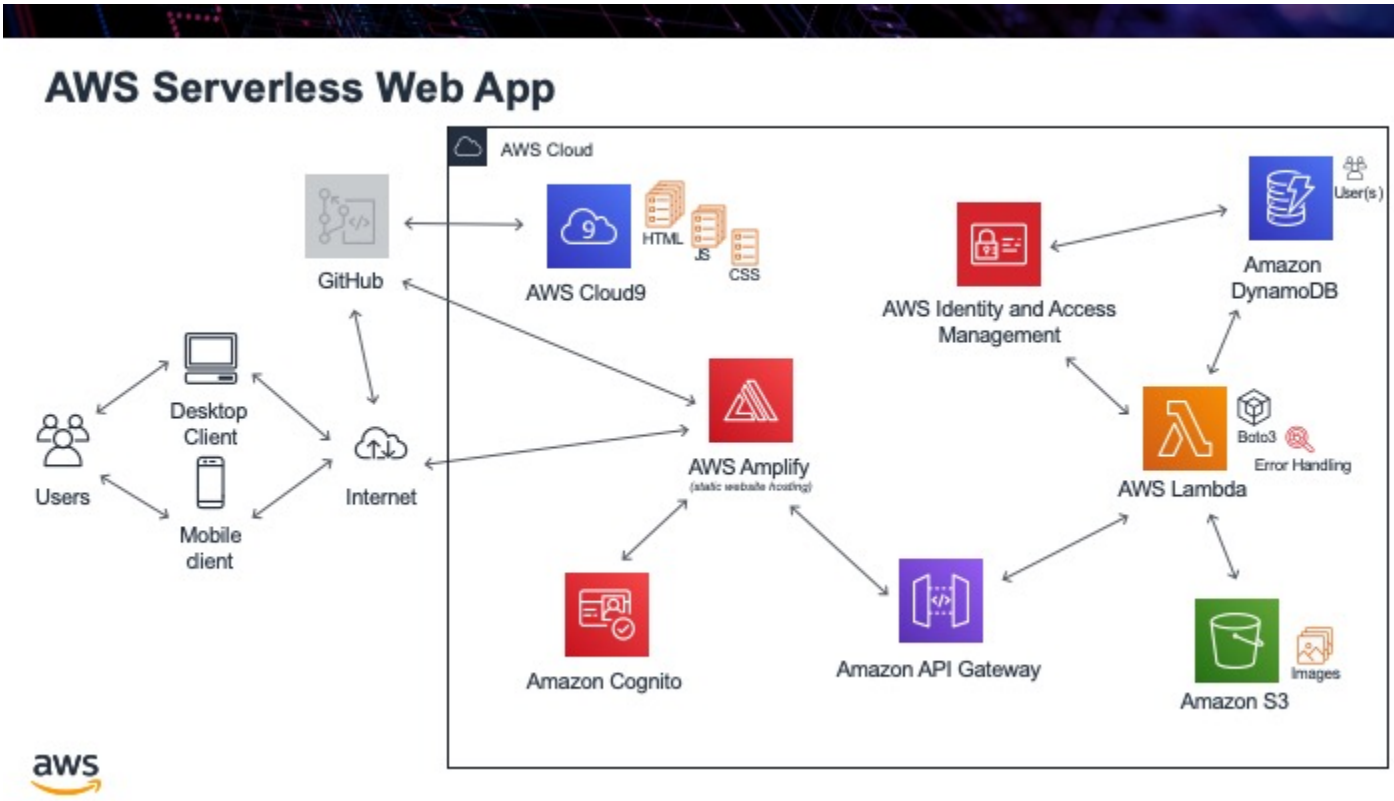
AWS Serverless Web Application

Patrick Coxall

Nov 17, 2020

Contents



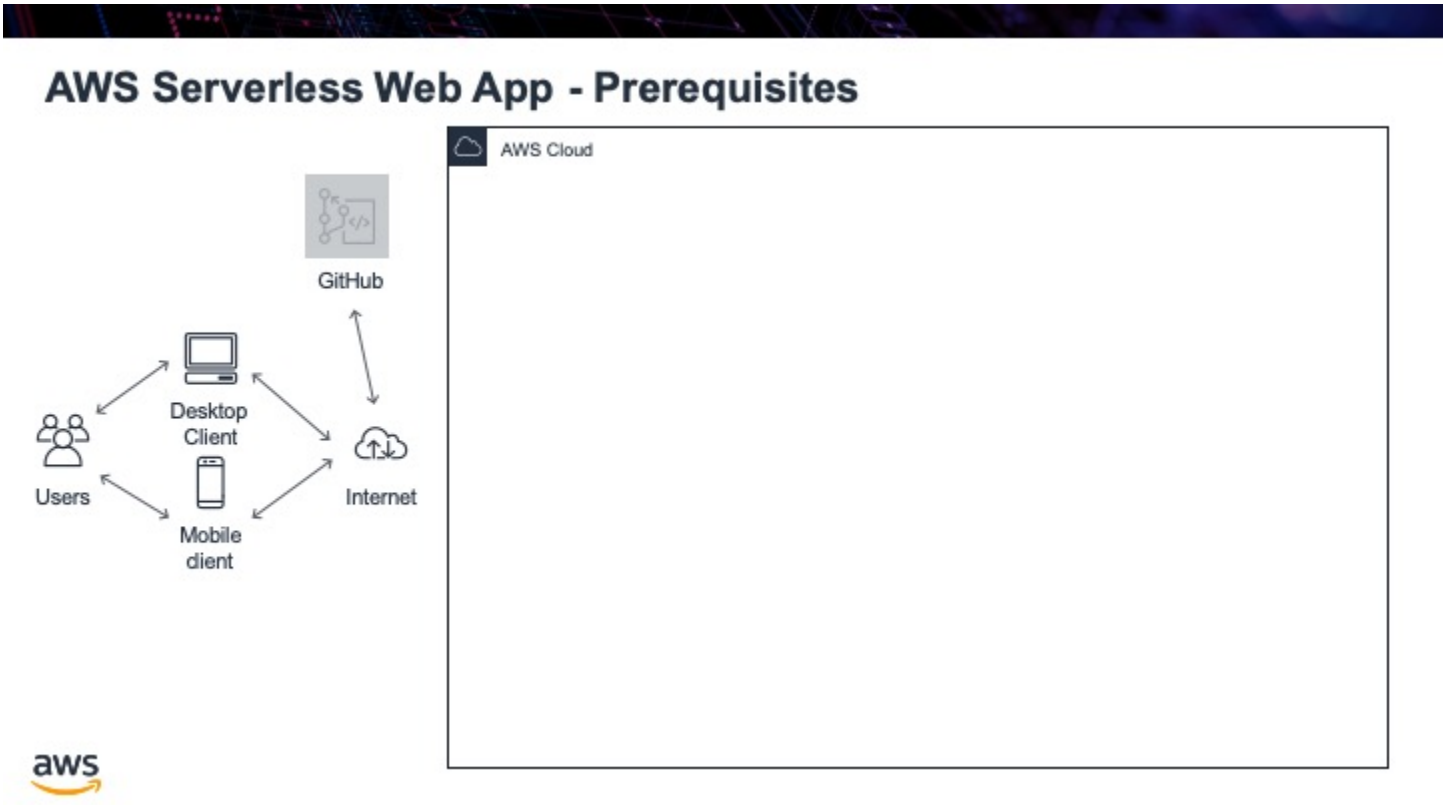
In this project we will be making an AWS Serverless Web App. We will be using an AWS Educate Student account, so anyone should have enough permissions to make this app. It is assumed that you already have not only an AWS account but also a GitHub account and know basic HTML, CSS, JavaScript and Python for the back end.

See also:

All the code can be found at this GitHub link.

CHAPTER 1

Prerequisites



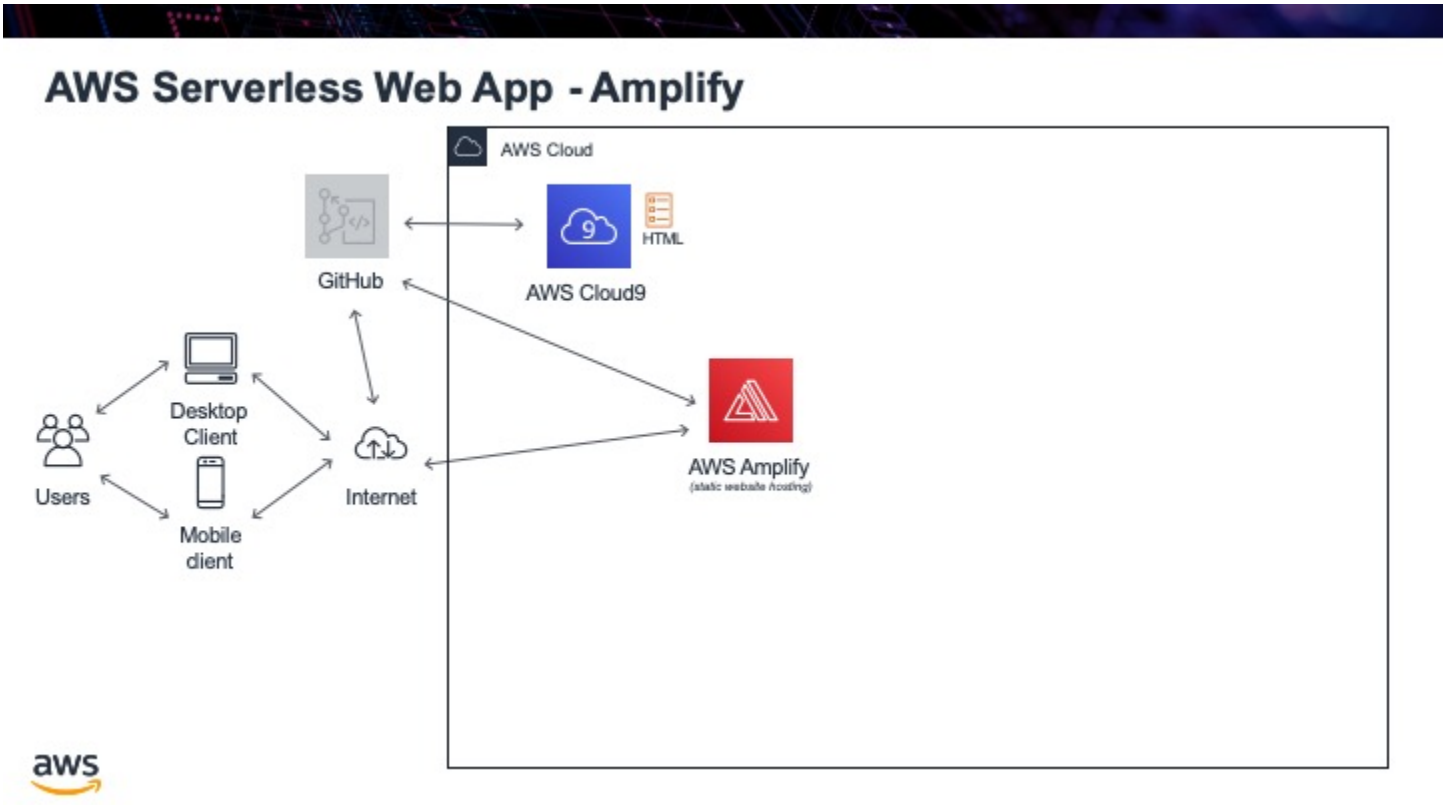
To do this tutorial, it is assumed that you have access to certain tools and that you know the basics of certain programming technologies. To be able to complete this tutorial you will need:

- Amazon Web Services (AWS) access. In the tutorial I will be using an AWS Education student account. It is more restrictive that a regular AWS account. All functionality in this tutorial is able to be done with a student account though.

- GitHub account. If you do not already have one, you can register for a free account.
- Basic HTML, CSS & JavaScript coding experience
- Basic JSON experience, since our Lambda function will be returning JSON as its output
- Baisc Python coding exerience, for doing the back end code in Lambda in Python

CHAPTER 2

Amplify



We will use AWS Amplify to host our web app so it is always available and we do not need to worry about provisioning servers, auto-scaling instances or anything else about managing web content. We will also use AWS Cloud9 as our IDE to write our HTML in and then publish all the code to GitHub as our version control system.

Tasks:

- create GitHub repo

- create Cloud9 instance, remember to delete the README.md file from CLoud9 since the GitRepo has one and you do not want any conflicts
- initialize git repo in Cloud9
- then change the default branch from "master" to "main" (since we no longer use the terminology master for repos)
- connect the root of Cloud9 instance to GitHub repo
- create `index.html` file in the root of Cloud9
- update GitHub repo
- create Amplify instance connected to GitHub repo
- ensure you select the "main" branch and not the "master" one we will not be using

Listing 1: How to connect Cloud9 instance root to GitHub repo

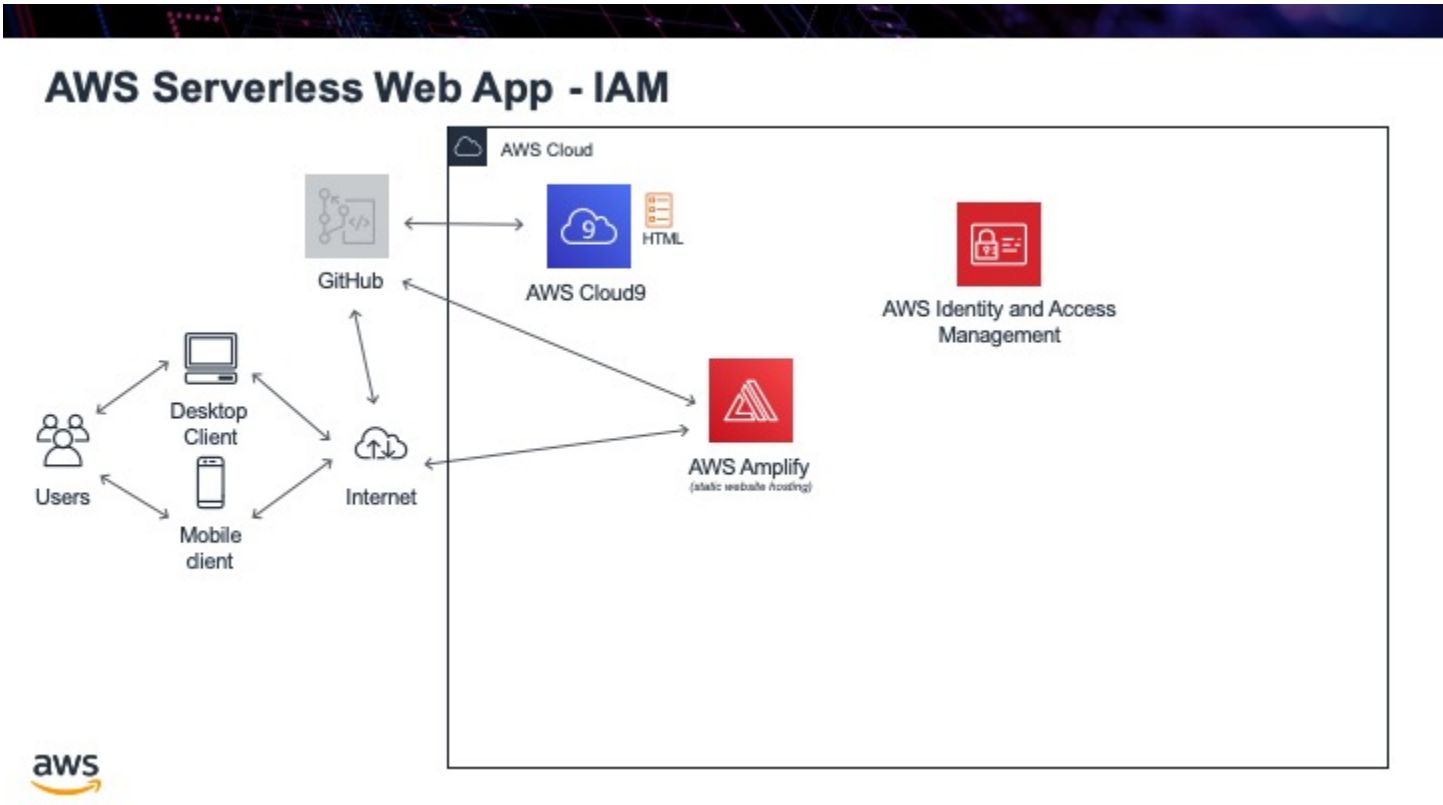
```
vocstartsoft:~/environment $ git init
Initialized empty Git repository in /home/ubuntu/environment/.git/
vocstartsoft:~/environment (master) $ git checkout -b main
vocstartsoft:~/environment (master) $ git remote add origin https://github.com/Mr-
↪Coxall/Amplify-Test
vocstartsoft:~/environment (master) $ git pull origin main
```

Listing 2: index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>AWS Serverless Web App</title>
  </head>
  <body>
    Hello, World!
  </body>
</html>
```

CHAPTER 3

IAM



Before we do any more work with AWS, the next thing we need is to create some permissions so we can give access to certain AWS services by other services. For example, eventually, we will want some code (Lambda function) to access the database. Unless we create permissions to let this happen, hopefully, the database will not just let anyone access it!

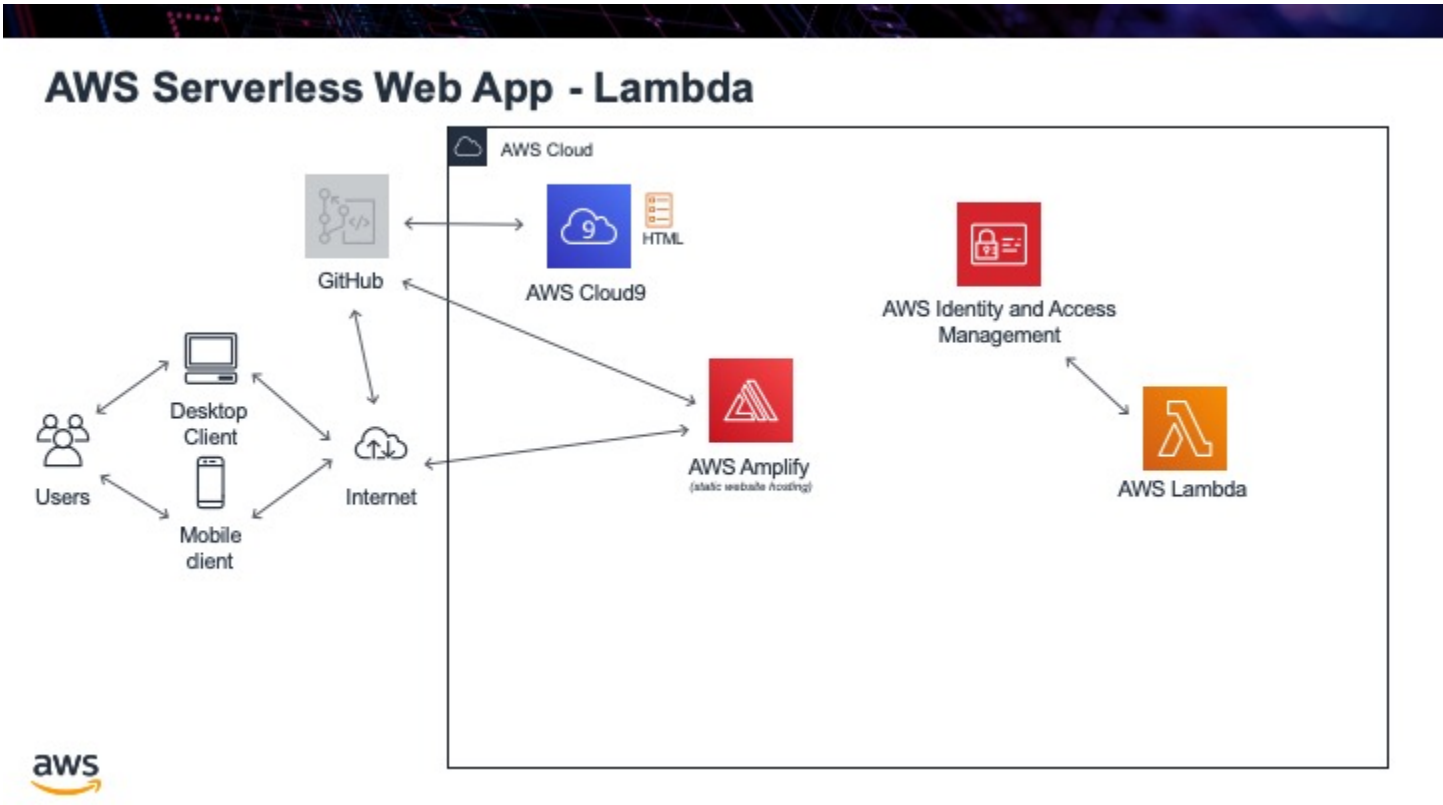
To do this, we will use AWS IAM (Identity and Access Management). We will create a "role" that will give Lambda functions permissions to access DynamoDB, the database we will be using.

Tasks:

- create a role in IAM, for Lambda to access DynamoDB

CHAPTER 4

Lambda Function



We will use AWS Lambda to run our back end code. The language we will use is Python, although other languages are available. Once again the advantage of running the code in AWS Lambda is that we need not worry about any provisioning and maintenance of servers, and their function will always be available. To start off with we will create just a simple “Hello, World!” program. Unlike normal programs where the output is for a user to see, the output from our Lambda function is for another program to use, so the output will be in a machine-friendly format, namely JSON.

Tasks:

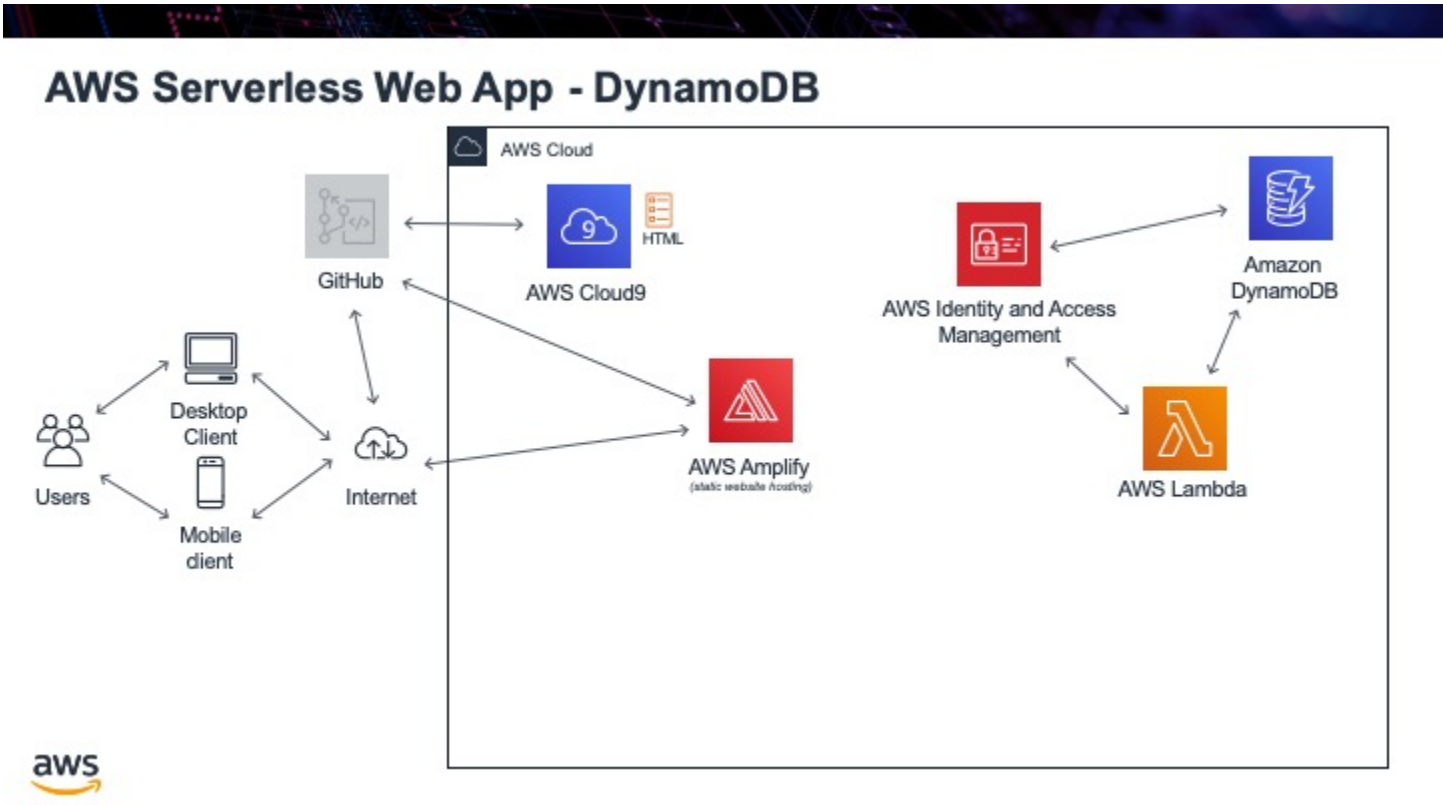
- create a Lambda function that returns "Hello, World!"
- we need to ensure we use the "role" we just created previously
- we also need to create "test cases" for it to use as input when we test the function to ensure it is running correctly
- to save our changes to see if the test cases will run correctly, you need to click "Deploy"

Listing 1: hello_world.py Lambda function

```
#!/usr/bin/env python3

# Created by: Mr. Coxall
# Created on: Jan 2020
# This function is the Hello, World! Lambda function

import json

def lambda_handler(event, context):
    # TODO implement

    return_var = {
        'statusCode': 200,
        'body': json.dumps('Hello, ' + event['name'])
    }

    return return_var
```

CHAPTER 5

DynamoDB



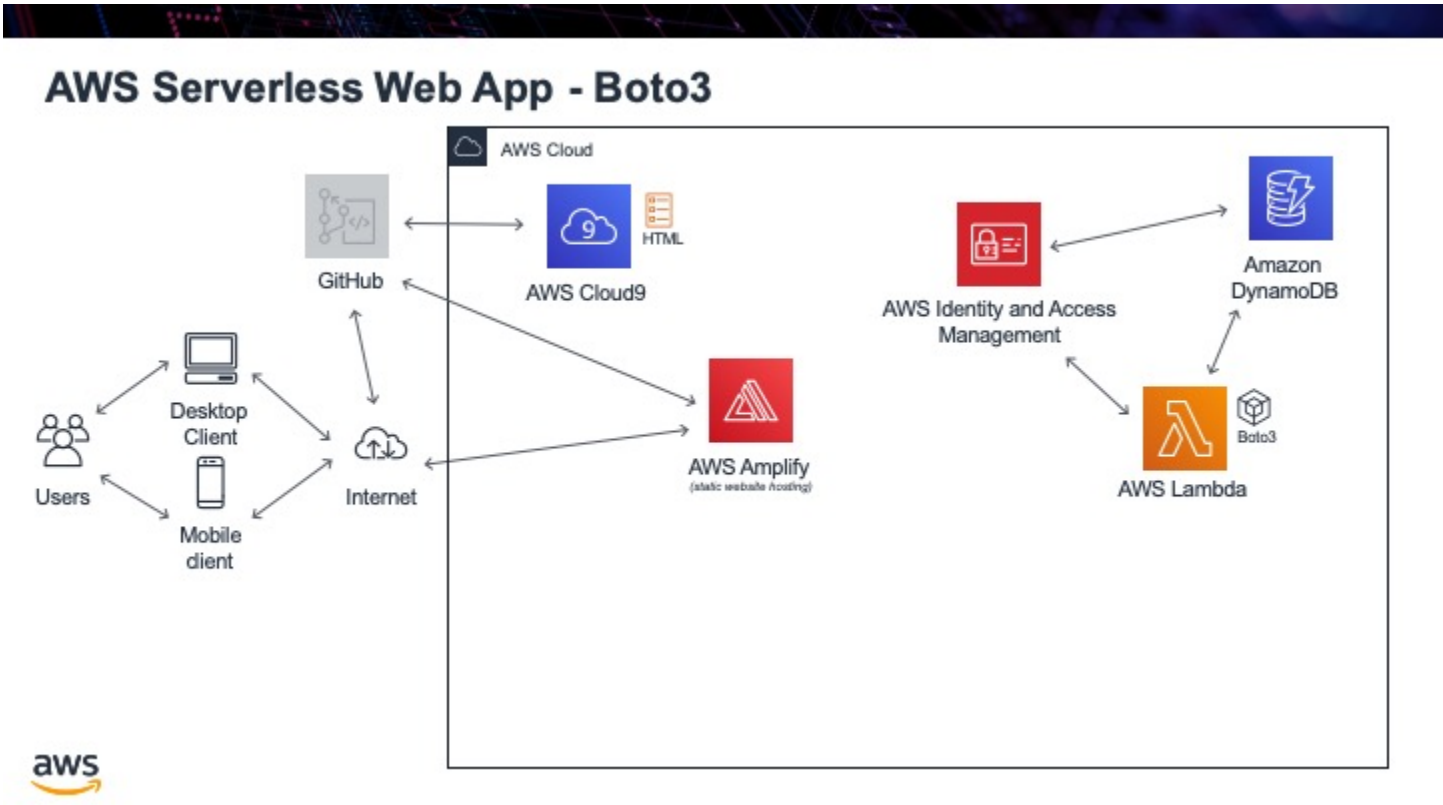
We will use AWS DynamoDB as our back-end database. This is a NoSQL database service, so if you have never used databases before, do not worry. Just assume that the table we create is just like a simple spreadsheet, it has rows with different information in it and it saves the information in columns. Just like a good spreadsheet, we need to have a unique "key" that uniquely identifies each row. In a spreadsheet, it usually has row numbers. In our example, we will use an email address, since that is a good unique key for a user of a web application.

Tasks:

- create a DynamoDB table
- we will place some data into our database by hand, just to ensure it works correctly
- change the "capacity" of the database because for our tests, we do not need so much power (and it will cost less!)

CHAPTER 6

Boto3



We will once again revisit Lambda functions. Now that we have a database with information in it, we can create a Lambda function that will retrieve a row (or several rows). As part of AWS, there is a Python library called Boto3. This library allows you to access any AWS service using Python code. We will use this library to access the DynamoDB, get back a JSON file, and then return it from Lambda.

There will be just one problem when we get back the data from DynamoDB. We created a row that was "age" and holds a "decimal" value type. It turns out that Boto3 returns this type of value in a *strange* format, that if we just

returned it from lambda as JSON we would get an error. To prevent this, we will include a helper function that will re-format our decimal number value into something we can use.

Tasks:

- create a new Lambda function
- add in the Boto3 library to access the database
- query the database to return a row of information
- return the row in JSON format from the lambda function
- we will pull out the "Item" element from the returned data since that is what we are interested in
- if no row was returned, then we need to catch this case and return an empty return statement
- if there is a row returned, we need to re-format the "decimal" type, so that we do not get an error. We will once again revisit Lambda functions. Now that we have a database with information in it, we can create a Lambda function that will retrieve a row (or several rows). As part of AWS, there is a Python library called Boto3. This library allows you to access any AWS service using Python code. We will use this library to access the DynamoDB, get back a JSON file and then return it from Lambda.

Listing 1: test case for Lambda function

```
{
  "email_address": "jane.smith@gmail.com"
}
```

Listing 2: get_user_info.py Lambda function

```
#!/usr/bin/env python3

# Created by: Mr. Coxall
# Created on: Jan 2020
# This function returns a row from our chocolate_user DynamoDB

import json
import boto3
import decimal


def replace_decimals(obj):
        # Helper class to Decimals in an arbitrary object
        #   from: https://github.com/boto/boto3/issues/369

    if isinstance(obj, list):
        for i in range(len(obj)):
            obj[i] = replace_decimals(obj[i])
        return obj
    elif isinstance(obj, dict):
        for k, v in obj.items():
            obj[k] = replace_decimals(v)
        return obj
    elif isinstance(obj, set):
        return set(replace_decimals(i) for i in obj)
    elif isinstance(obj, decimal.Decimal):
        if obj % 1 == 0:
            return int(obj)
        else:
```

(continues on next page)

(continued from previous page)

```
            return float(obj)
    else:
        return obj


def lambda_handler(event, context):
    # get a row from our chocolates_user table

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('chocolate_users')
    response = table.get_item(
        Key = {
            'email':event['email_address']
        }
    )

    try:
        results = response["Item"]
        results = replace_decimals(results)
    except:
        results = {}

    return {
        'statusCode': 200,
        'body': json.dumps(results)
    }
```

See also:

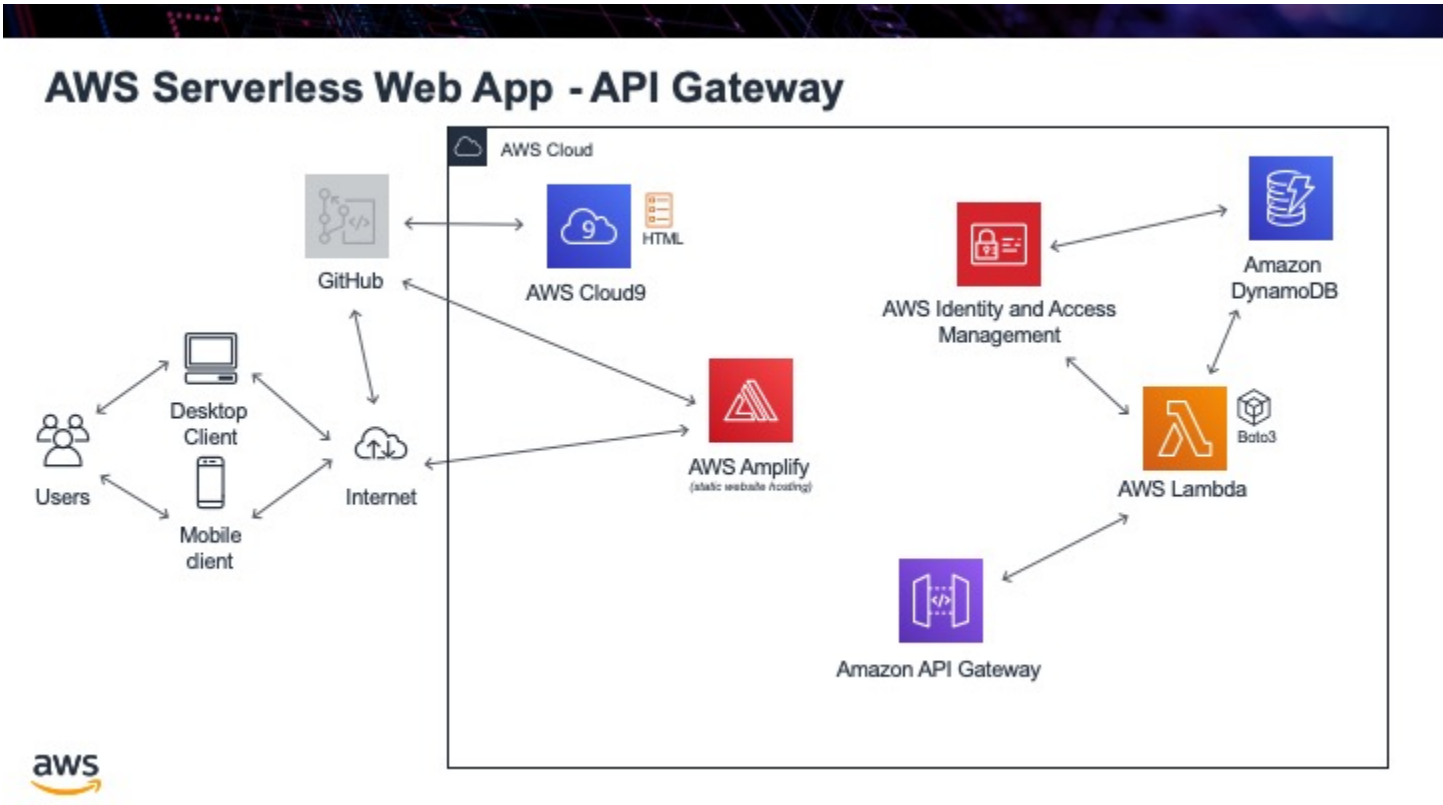
Boto3 documentation for DynamDB

See also:

DynamDB Python examples

CHAPTER 7

API Gateway



Now that we have a Lambda function that will access our table and return information, we need to make that logic accessible on the Internet. This way we can write HTML and JavaScript code to get the information and present it on a web page. To do this, we will use AWS API Gateway.

API Gateway is a service that can make a Lambda function accessible by a URL. Just like our Lambda function needed a parameter passed to it to know what user information to get back, as part of the URL we will pass in what user information we would like back.

Tasks:

- create a new API Gateway
- add in CORS, so that any URL can access our API
- create a "GET" request, to get the user info
- add in a "mapping" template, to specify what parameters it allows to be passed in
- Enable CORS, or we cannot access the API due to being in different domains
- publish the API, so it is visible on the Internet

Listing 1: Mapping Templates

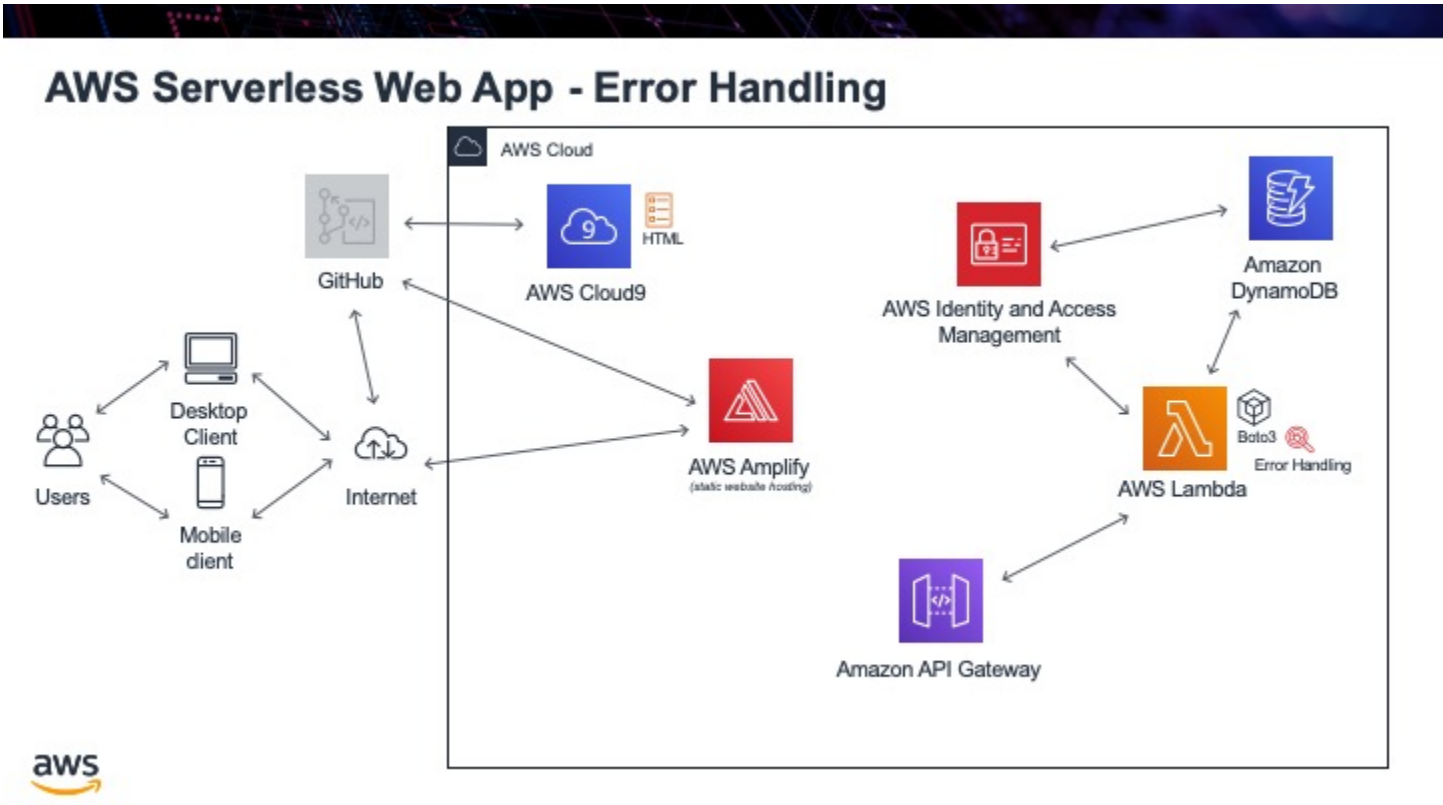
```
## set what the Lambda parameter is named : what name will be passed in the URL

#set($inputRoot = $input.path('$'))
{"email_address":"$input.params('user_email')"}
```

Listing 2: Query Strings for testing

```
user_email=jane.smith@gmail.com
```

CHAPTER 8

Error Handling



Our API Gateway is now working if you pass the correct parameters into it. The next question is what happens if someone does not pass the proper parameters in or even no parameters at all? Sadly, we get a really nasty error message coming back. This is OK, but really we should be nice programmers and handle it better. What we will do is alter our Lambda code to trap these kinds of errors.

Tasks:

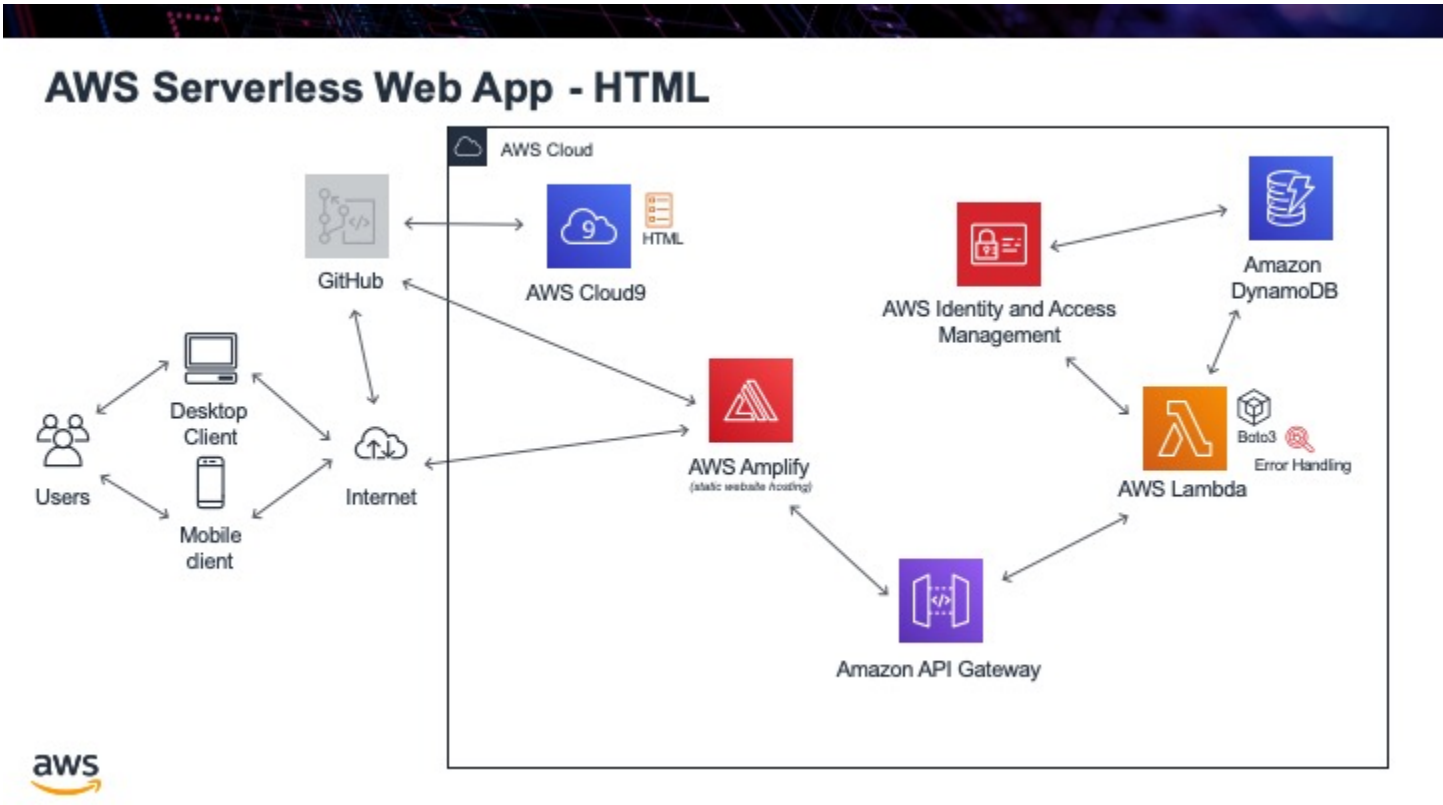
- use try and except to catch wrong or no parameters being passed

Listing 1: get_user_info.py Lambda function, with error handling

```
def lambda_handler(event, context):
    # function returns a row from our chocolate_user DynmamoDB

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('chocolate_user')

    try:
        response = table.get_item(
            Key = {
                'email':event['email_address']
            }
        )

        try:
            result = response['Item']
            result = replace_decimals(result)
        except:
            result = {}

        print(result)

        return_var = {
            'statusCode': 200,
            'body': json.dumps(result)
        }

        return return_var

    except:
       return {
            'statusCode': 204,
            'body': json.dumps({})
        }
```

CHAPTER 9

HTML



Now that we have a fully functional API, let's write some HTML and JavaScript to show the information on a webpage. We will use the JavaScript "Fetch" method to call our API, get back the JSON file, parse out the information we want and then present that on our webpage inside a element.

Tasks:

- add a `<div>` section to hold the data

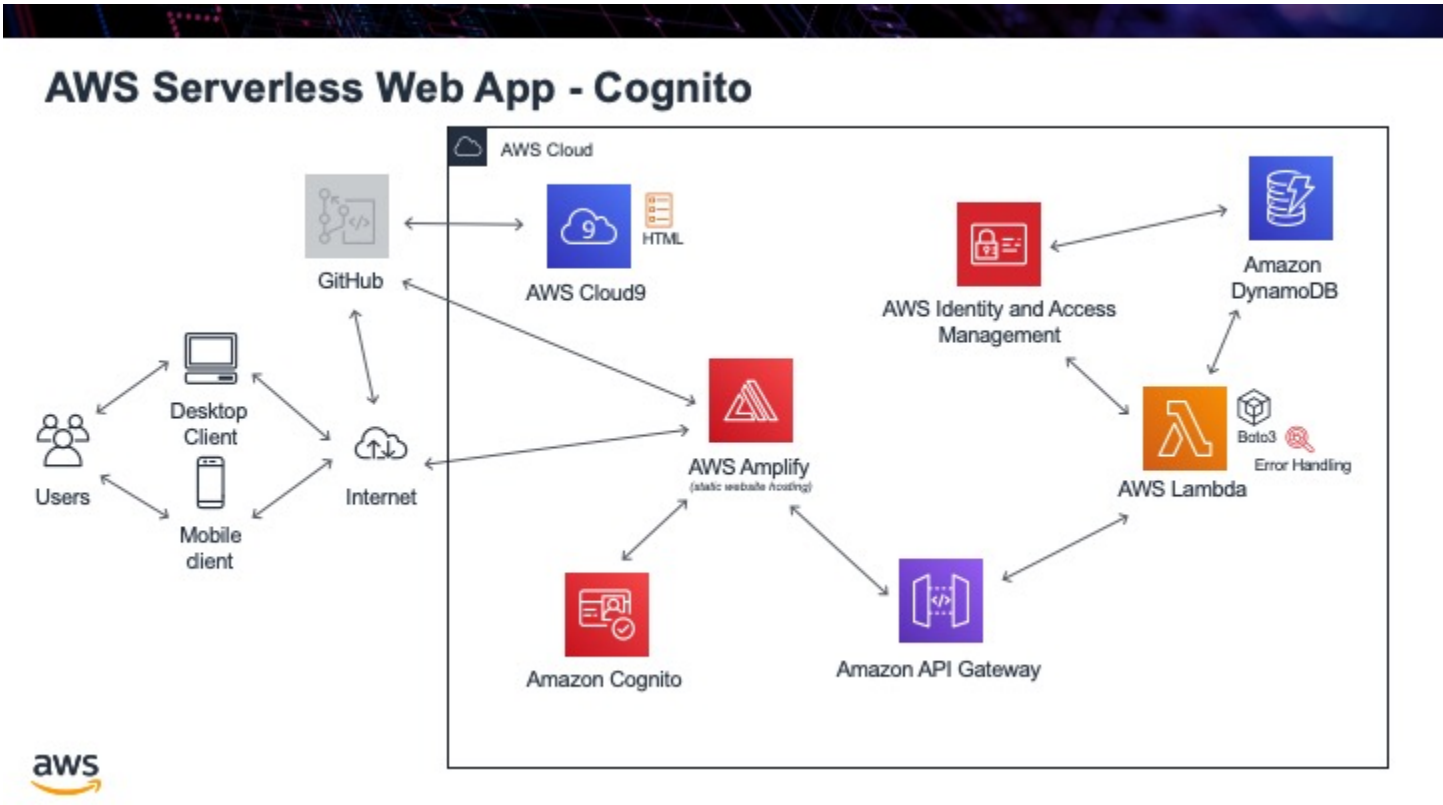
- add a `<script>` section, calling our API
- get back JSON file and place the info in a variable
- present the data in our `index.html` file

Listing 1: index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web App</title>
  </head>
  <body>
    <div id="user_info">
      <p>Please wait ...</p>
    </div>
    <div>
      <button onclick="getUser()">Click me</button>
    </div>
  </body>

  <script type="text/javascript">

    async function getUser() {
      // get the user info from API

      const api_url = 'your_API_URL?user_email=jane.smith@gmail.com'
      const api_response = await fetch(api_url);
      const api_data = await(api_response).json();

      const div_user_info = document.getElementById('user_info');
      div_user_info.innerHTML = api_data['body'];
    }
  </script>
</html>
```

CHAPTER 10

Cognito



The next major task is to have the ability to sign in to our website. This prevents people from accessing things they should not and will also give us the ability to present individualized content for a particular user that has signed in since we know who is looking at the website. So instead of us hard coding the user email address to get back the information from the database, since a user has signed in, it should automatically return their information.

To do this, we will use an AWS service called Cognito. This will provide all user account management; importantly managing the user's login and password. You *do not want to manage the user's passwords yourself*, this is a really,

really complicated thing to do properly (see the video below on why you should never save user passwords yourself). To avoid this, AWS Cognito will manage our user login, the email address, and the passwords.

Tasks:

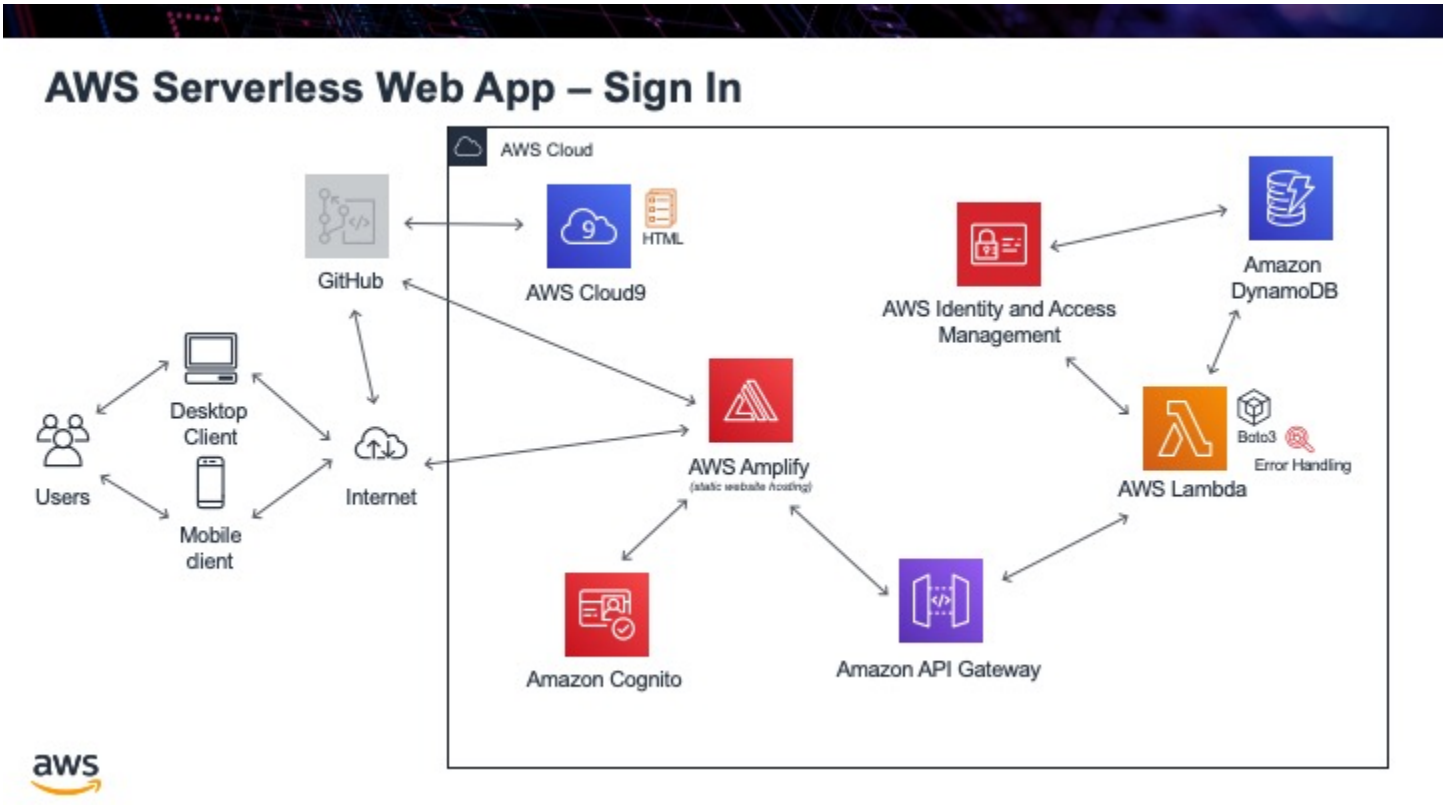
- create an AWS Cognito user pool
- create an app client
- use the Cognito built-in signup URL to create a user and click on the provided link to confirm the user
- confirm the user now exists in the Congnito pool
- ensure this user also exists in the DynamoDB table, so that this user has information in your table

See also:

Most of my Cognito code came from this tutorial

CHAPTER 11

Sign In



Now that we have a user in the Cognito system, we need a way for the user to sign in to the app. Cognito provides a built-in UI for this, but it does not work nicely with the way we will build our app. We will write our own HTML page, that uses some provided JavaScript libraries to log in.

What we will do is download the libraries we need and place them in a folder called "js". This is a standard name used to place all your JavaScript files in. Once all the libraries are downloaded, you then need to update the `config.js` file with the information from Cognito. Then we will write the HTML to have the login and password entered and a

button to log in. When you click on the button, we will call a function to check if this is a valid user from Cognito and if it is, we will return a **token** and the username. This will prove it is a valid user.

Tasks:

- download the JavaScript libraries, from here, and place them in a js folder
- update the `config.js` file with your app information
- write file `sign-in.html`, that has 2 input boxes and a sign-in button
- write the JavaScript function to sign the user in
- if the user is valid, show the user email address, returned from Cognito and a JWT token

Listing 1: config.js

```
window._config = {
  cognito: {
    userPoolId: 'xxx', // e.g. us-east-1_uXboG5pAb
    region: 'us-east-1', // e.g. us-east-1
    clientId: 'yyy' // e.g. 6m2mqsko56fo558pp9g54ht4pb
  },
};
```

Listing 2: sign-in.html

```
<!DOCTYPE html>

<html lang="en">
  <head>
  <meta charset="utf-8">

    <!-- Javascript SDKs-->
    <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
    <script src="js/amazon-cognito-auth.min.js"></script>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.596.0.min.js"></script>
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
  </head>

  <body>
    <form>
      <h1>Please sign in</h1>

      <input type="text" id="inputUsername"  placeholder="Email address" name=
↪"username" required autofocus>
      <input type="password" id="inputPassword"  placeholder="Password" name="password
↪" required>
      <button type="button" onclick="signInButton()">Sign in</button>
    </form>

    <br>
    <div id='logged-in'>
      <p></p>
    </div>

    <p>
      <a href="./profile.html">Profile</a>
    </p>
```

(continues on next page)

(continued from previous page)

```
    <br>
    <div id='home'>
      <p>
        <a href='./index.html'>Home</a>
      </p>
    </div>

    <script>

      var data = {
        UserPoolId : _config.cognito.userPoolId,
        ClientId : _config.cognito.clientId
      };
      var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
      var cognitoUser = userPool.getCurrentUser();

      function signInButton() {
        // sign-in to AWS Cognito

        var authenticationData = {
          Username : document.getElementById("inputUsername").value,
          Password : document.getElementById("inputPassword").value,
        };

        var authenticationDetails = new AmazonCognitoIdentity.
↪AuthenticationDetails(authenticationData);

        var poolData = {
          UserPoolId : _config.cognito.userPoolId, // Your user pool id here
          ClientId : _config.cognito.clientId, // Your client id here
        };

        var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);

        var userData = {
          Username : document.getElementById("inputUsername").value,
          Pool : userPool,
        };

        var cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);

        cognitoUser.authenticateUser(authenticationDetails, {
            onSuccess: function (result) {
              var accessToken = result.getAccessToken().getJwtToken();
              console.log(result);

              //get user info, to show that you are logged in
              cognitoUser.getUserAttributes(function(err, result) {
                  if (err) {
                    console.log(err);
                    return;
                  }
                  console.log(result);
                  document.getElementById("logged-in").innerHTML = "You are logged in␣
↪as: " + result[2].getValue();
              });
```

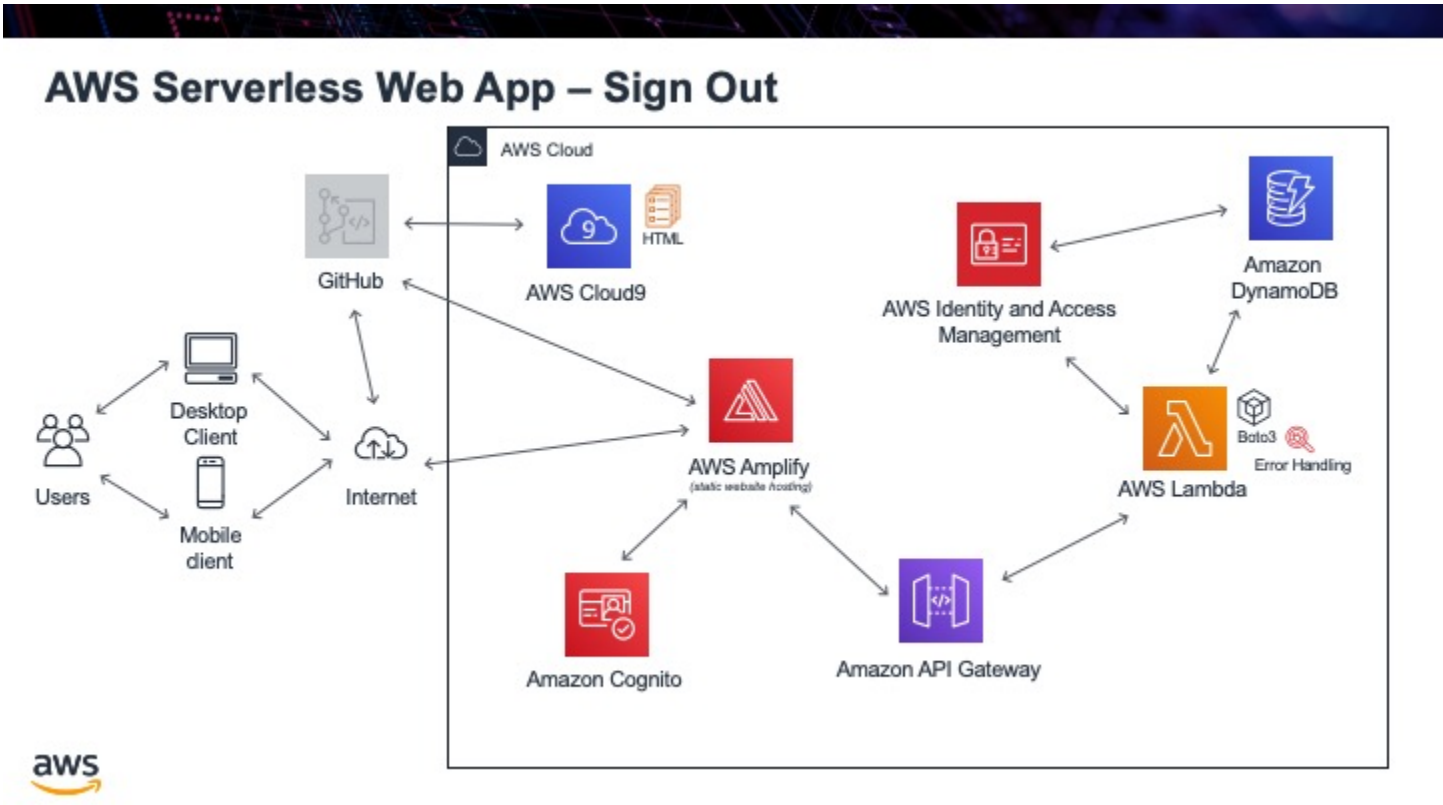
(continues on next page)

(continued from previous page)

```
87
88              },
89              onFailure: function(err) {
90                alert(err.message || JSON.stringify(err));
91              },
92          });
93        }
94      </script>
95
96    </body>
97  </html>
```

CHAPTER 12

Sign Out



Users can now sign in to our app and we can return their email address. The next logical step is to allow users to log out of the app, so someone else can not just get access to their data. To log out, we will use the same Cognito libraries, we will literally just call the **signout()** method and this will sign the user out. We will not use a button but call the JavaScript function as soon as the webpage loads.

Tasks:

- write file `sign-out.html`, that just has some text explaining you are now signed out
- write the JavaScript function to sign the user out
- confirm they actually sign the user in, and write this to the console just to prove it is working, then call **signout()** function

Listing 1: sign-out.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <!--Cognito JavaScript-->
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
  </head>

  <body>
  <div class="container">
    <div>
      <h1>Sign Out</h1>
      <p>Successfully signed-out</p>
    </div>

    <br>
    <div id='home'>
      <p>
      <a href='./index.html'>Home</a>
      </p>
    </div>
  </div>

  <script>
    var data = {
      UserPoolId : _config.cognito.userPoolId,
      ClientId : _config.cognito.clientId
    };
    var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
    var cognitoUser = userPool.getCurrentUser();

    window.onload = function(){
      if (cognitoUser != null) {
        cognitoUser.getSession(function(err, session) {
            if (err) {
              alert(err);
              return;
            }
            console.log('session validity: ' + session.isValid());

            // sign out
            cognitoUser.signOut();
            console.log("Signed-out");
        });
      } else {
        console.log("Already signed-out")
      }
    }
  </script>
```

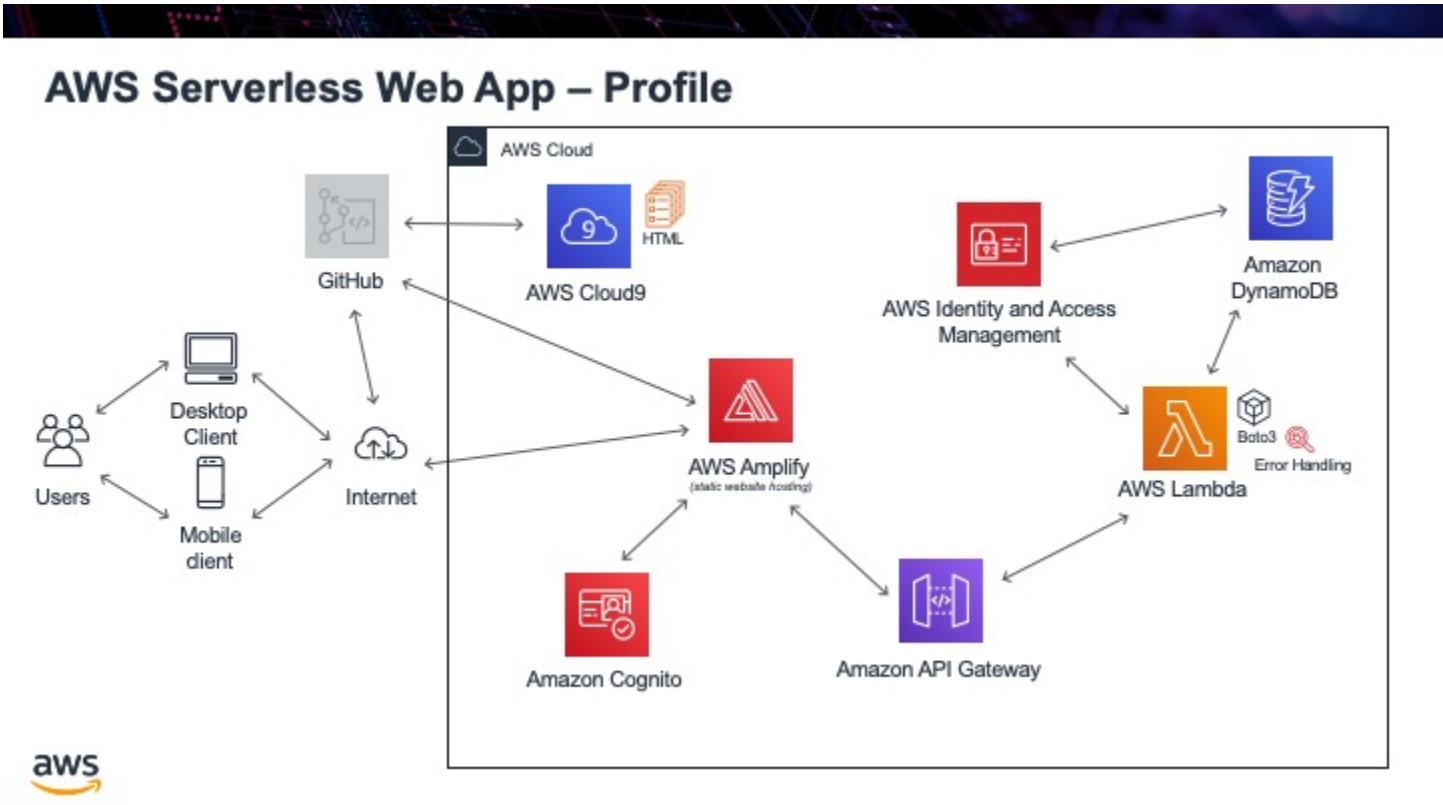
(continues on next page)

(continued from previous page)

```
51
52     </body>
53 </html>
```

CHAPTER 13

Profile



The next big step is to confirm that we can integrate what we have in `temp.html`, the ability to get user information from the database, with our ability to sign in a user. What we want is that after the user has signed in, go to a web page and automatically call the API to show the user info from the database.

To do this, we will take the `sign-out.html` code and copy it. We will remove the function that signs the user out, but keep the bit that confirms we sign the user in. Once we confirm we sign the user in, we will copy the function from `sign-in.html` that returns the user's email address. We will call this function, grab the email address, and pass it

as a parameter into the function from temp.html that calls the API. This will then hopefully return the user info from the database.

Tasks:

- copy `sign-out.html`
- remove sign-out code
- copy over **getUserAttributes()** function from `sign-in.html`
- copy over **getUser()** function from `temp.html`
- show profile results in div

Listing 1: profile.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <!--Cognito JavaScript-->
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
  </head>

  <body>
  <div class="container">
    <div>
      <h1>Profile</h1>
    </div>
    <div id='profile'>
      <p></p>
    </div>
  <div>

    <br>
    <div id='home'>
      <p>
        <a href='./index.html'>Home</a>
      </p>
    </div>

  <script>

    async function getUser(email_address) {
      // get the user info from API Gate

      const api_url = 'https://gonvpjbyuf.execute-api.us-east-1.amazonaws.com/prod/
↪user-profile?user_email=' + email_address;
      const api_response = await fetch(api_url);
      const api_data = await(api_response).json();
      console.log(api_data);

      const div_user_info = document.getElementById('profile');
      div_user_info.innerHTML = api_data['body'];
      }

    var data = {
      UserPoolId : _config.cognito.userPoolId,
```

(continues on next page)

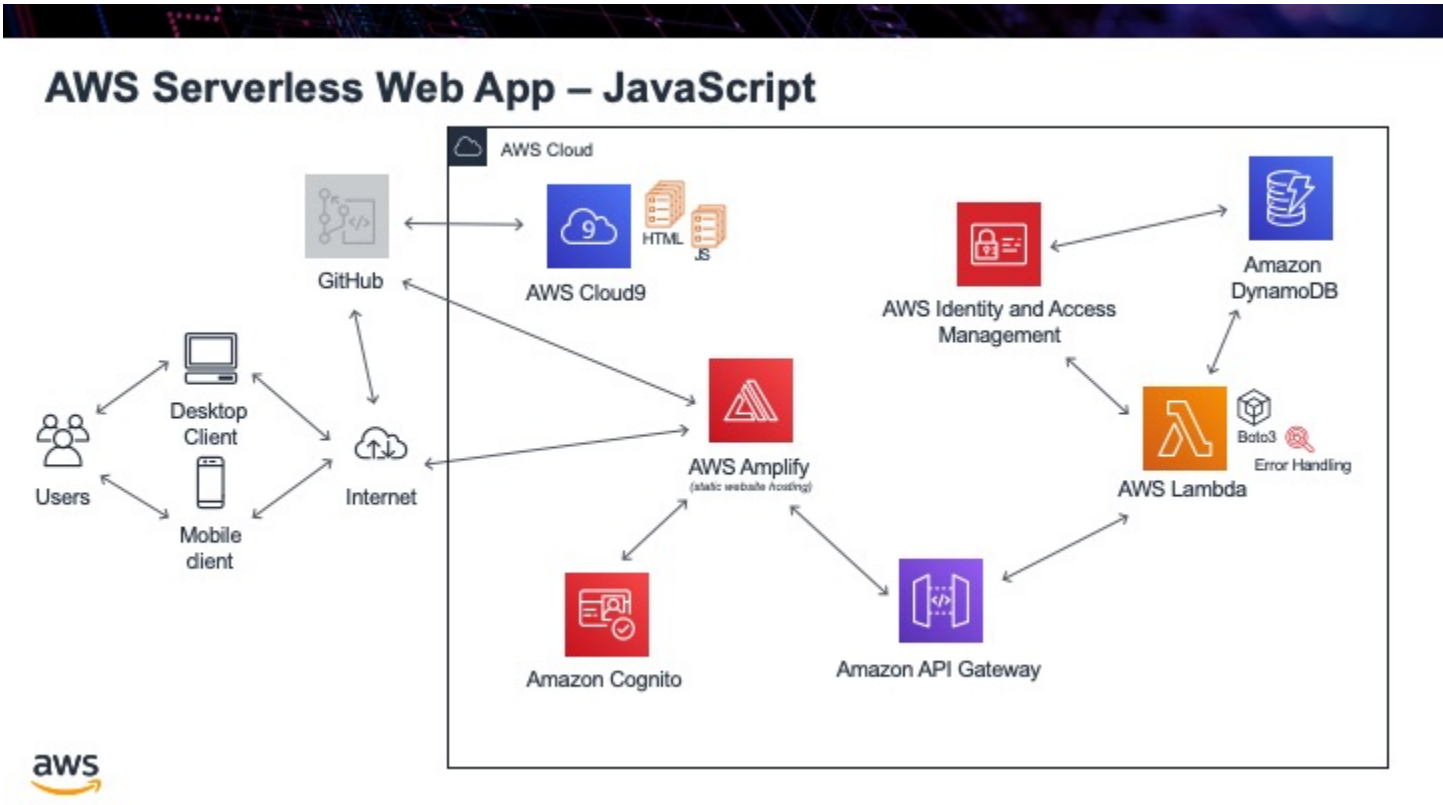
(continued from previous page)

```
        ClientId : _config.cognito.clientId
      };
      var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
      var cognitoUser = userPool.getCurrentUser();

      window.onload = function(){
        if (cognitoUser != null) {
          cognitoUser.getSession(function(err, session) {
            if (err) {
              alert(err);
              return;
            }
            //console.log('session validity: ' + session.isValid());

            cognitoUser.getUserAttributes(function(err, result) {
              if (err) {
                console.log(err);
                return;
              }
              // user email address
              console.log(result[2].getValue());
              getUser(result[2].getValue())
            });

          });
        } else {
          console.log("Already signed-out")
        }
      }
    </script>

  </body>
</html>
```

Warning: The above code is a **really** bad way to write this. Anyone can look at the html and see the line: `const api_url = 'https://gonvpjbyuf.execute-api.us-east-1.amazonaws.com/prod/user-profile?user_email=' + email_address;` and then just place any email address they want in. Once they find a valid email address, they will then get full access to all the user's info. I know it is un-secure but for now we are going to leave it .

CHAPTER 14

JavaScript



The basics of our app are now complete. The next thing to tackle is to fix up our HTML. It is good practise to actually remove the JavaScript code from within the HTML documents and place it in a seperate file. This way the code and content are seperate. What we will do to start is remove the **get_user()** function from `profile.html` and call it from our HTML. We will then remove the JavaScript code from `sign-in.html` and `sign-out.html` and create `*.js` files for the JavaScript code in each of these files. When done, all JavaScript will be moved out of the HTML files and into their own files.

Tasks:

- create a js directory and a JavaScript file for our code
- move the **<script>** code from `sign-in.html` into a `sign-in.js` file
- fix the code up, so that there are no **global** variables, since all code must now be in functions
- call the new function from `sign-in.html`
- do the same process to `sign-out.html` and `profile.html`

14.1 Sign In Code

sign-in.html

sign-in.js

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">

    <!-- Javascript SDKs-->
    <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
    <script src="js/amazon-cognito-auth.min.js"></script>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.596.0.min.js"></script>
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
    <script type="text/javascript" src="js/sign-in.js"></script>
  </head>

  <body>
    <form>
      <h1>Please sign in</h1>

      <input type="text" id="inputUsername"  placeholder="Email address" name=
↪"username" required autofocus>
      <input type="password" id="inputPassword"  placeholder="Password" name="password
↪" required>
      <button type="button" onclick="signInButton();">Sign in</button>
    </form>

    <br>
    <div id='logged-in'>
      <p></p>
    </div>

    <p><a href="./profile.html">Profile</a></p>

    <br>
    <div id='home'>
      <p>
        <a href='./index.html'>Home</a>
      </p>
    </div>

```

(continues on next page)

(continued from previous page)

```
  </body>
</html>
```

```
// JavaScript File

function signInButton() {
  // sign-in to AWS Cognito

  var data = {
      UserPoolId : _config.cognito.userPoolId,
    ClientId : _config.cognito.clientId
  };
  var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
  var cognitoUser = userPool.getCurrentUser();

    var authenticationData = {
    Username : document.getElementById("inputUsername").value,
    Password : document.getElementById("inputPassword").value,
  };

  var authenticationDetails = new AmazonCognitoIdentity.
↪AuthenticationDetails(authenticationData);

  var poolData = {
    UserPoolId : _config.cognito.userPoolId, // Your user pool id here
    ClientId : _config.cognito.clientId, // Your client id here
  };

  var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);

  var userData = {
    Username : document.getElementById("inputUsername").value,
    Pool : userPool,
  };

  var cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);

  cognitoUser.authenticateUser(authenticationDetails, {
    onSuccess: function (result) {
      var accessToken = result.getAccessToken().getJwtToken();
      console.log(result);

      //get user info, to show that you are logged in
            cognitoUser.getUserAttributes(function(err, result) {
                if (err) {
                    console.log(err);
                    return;
                }
                console.log(result);
                document.getElementById("logged-in").innerHTML = "You are logged in␣
↪as: " + result[2].getValue();

                // now auto redirect to profile page
                window.location.replace("./profile.html");
            });

```

(continues on next page)

(continued from previous page)

```
    },
    onFailure: function(err) {
      alert(err.message || JSON.stringify(err));
    },
  });
}
```

14.2 Sign Out Code

sign-out.html

sign-out.js

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <!--Cognito JavaScript-->
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
    <script src="js/sign-out.js"></script>
  </head>

  <body>
    <div class="container">
      <div>
        <h1>Sign Out</h1>
        <div id='sign-out'>
          <p>One moment please ...</p>
        </div>
      </div>
    <div>

    <br>
    <div id='home'>
      <p>
        <a href='./index.html'>Home</a>
      </p>
    </div>
  </body>
  <script>
    window.onload = function(){
      const temp_var = signOut();
    }
  </script>
</html>
```

```
// JavaScript File

function signOut() {
  //

  return_message = "";

```

(continues on next page)

(continued from previous page)

```
  const data = {
    UserPoolId : _config.cognito.userPoolId,
    ClientId : _config.cognito.clientId
  };
  const userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
  const cognitoUser = userPool.getCurrentUser();

  if (cognitoUser != null) {
    cognitoUser.getSession(function(err, session) {
      if (err) {
        alert(err);
        return;
      }
      console.log('session validity: ' + session.isValid());

      // sign out
      cognitoUser.signOut();
      console.log("Signed-out");
      return_message = "Signed-out";
    });
  } else {
    console.log("Already signed-out")
    return_message = "Already signed-out";
  }

  const div_user_info = document.getElementById('sign-out');
  div_user_info.innerHTML = return_message;
}
```

14.3 Profile Code

profile.html

profile.js

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <!--Cognito JavaScript-->
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
    <script type="text/javascript" src="js/profile.js"></script>
  </head>

  <body>
    <div class="container">
      <div>
        <h1>Profile</h1>
      </div>
      <div id='profile'>
        <p>One moment please ...</p>
      </div>
    <div>
```

(continues on next page)

(continued from previous page)

```
    <br>
    <div id='home'>
      <p>
        <a href='./index.html'>Home</a>
      </p>
    </div>
  </body>
  <script>
    window.onload = function(){
      const temp_var = getUserAttributes();
    }
  </script>
</html>
```

```
// JavaScript File

async function getUser(email_address) {
    // get the user info from API Gate

    const api_url = 'https://gonvpjbyuf.execute-api.us-east-1.amazonaws.com/prod/user-profile?user_email=' + email_address;
    const api_response = await fetch(api_url);
    const api_data = await(api_response).json();
    console.log(api_data);

    const div_user_info = document.getElementById('profile');
    div_user_info.innerHTML = api_data['body'];
  }

function getUserAttributes() {
  var data = {
    UserPoolId : _config.cognito.userPoolId,
    ClientId : _config.cognito.clientId
  };
  var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
  var cognitoUser = userPool.getCurrentUser();

  if (cognitoUser != null) {
    cognitoUser.getSession(function(err, session) {
      if (err) {
        alert(err);
        return;
      }
      //console.log('session validity: ' + session.isValid());

      cognitoUser.getUserAttributes(function(err, result) {
        if (err) {
          console.log(err);
          return;
        }
        // user email address
        console.log(result[2].getValue());
        getUser(result[2].getValue())
      });

```

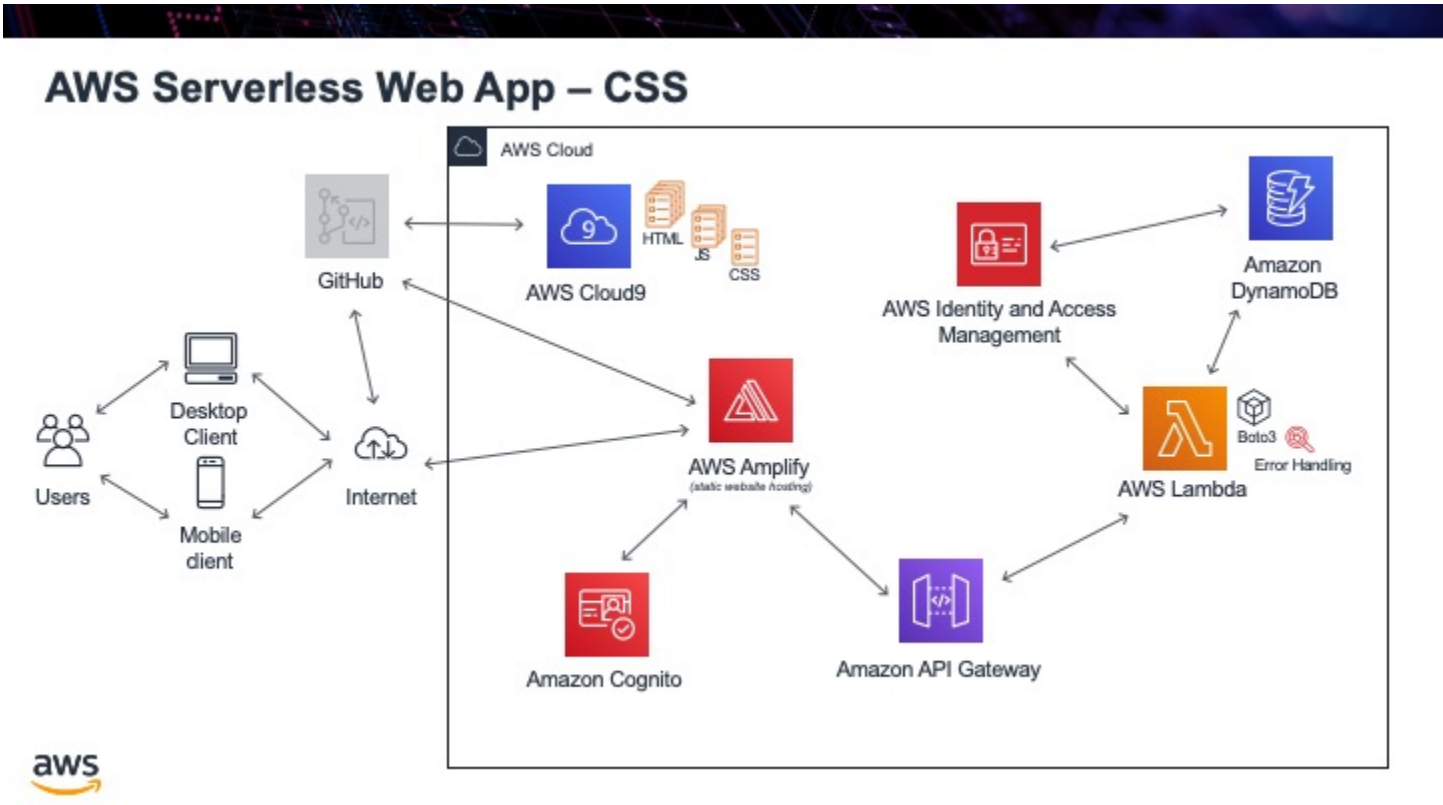
(continues on next page)

(continued from previous page)

```
    });
  } else {
    console.log("Already signed-out")
  }
}
```

CHAPTER 15

CSS



To make our website look more professional and also be help in maintaining its style, we will be using Google's MDL. This will act as CSS to our website. Most basic websites have a navigation menu at the top and the content below. We will use this model. The navigation menu will contain:

- home
- sign up

- sign in (and will auto move you to profile page)
- sign out
- profile

To use Google's MDL, you add 2 CSS references in the **head** of your HTML code. MDL is based on components. Components can be things like a button, or a navigation menu system, or a form. I will add these components to the HTML to create a better looking website.

Tasks:

- fix up some HTML code
- add Google's MDL CSS from its CDN, so we do not have to host it

15.1 HTML

Home

Products

Sign In

Sign Out

About

Profile

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="AWS Serverless Web App tutorial">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="Mr. Coxall">
    <meta name="date" content="Jan 2020">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--Google's MDL-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
↪family=Material+Icons">
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.brown-indigo.
↪min.css" />
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>

    <title>Chocolates</title>
  </head>

  <body>
    <!-- Always shows a header, even in smaller screens. -->
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
      <header class="mdl-layout__header">
        <div class="mdl-layout__header-row">
          <!-- Title -->
          <span class="mdl-layout-title">Chocolates</span>
          <!-- Add spacer, to align navigation to the right -->
          <div class="mdl-layout-spacer"></div>
          <!-- Navigation. We hide it in small screens. -->
```

(continues on next page)

(continued from previous page)

```
          <nav class="mdl-navigation mdl-layout--large-screen-only">
            <a class="mdl-navigation__link" href="./index.html">Home</a>
            <a class="mdl-navigation__link" href="./products.html">Product</a>
            <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
            <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
            <a class="mdl-navigation__link" href="./about.html">About</a>
          </nav>
        </div>
      </header>
      <div class="mdl-layout__drawer">
        <span class="mdl-layout-title">Chocolates</span>
        <nav class="mdl-navigation">
          <a class="mdl-navigation__link" href="./index.html">Home</a>
          <a class="mdl-navigation__link" href="./products.html">Product</a>
          <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
          <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
          <a class="mdl-navigation__link" href="./about.html">About</a>
        </nav>
      </div>
      <main class="mdl-layout__content">
        <div class="page-content"><!-- Your content goes here --></div>
      </main>
    </div>

    <!--Get below the nav menu at the top-->
    <br><br><br>

                <div class="container">
      <div>
        <h2>Welcome</h2>
        <div id='home'>
          <p></p>
        </div>
      </div>
                </div>

                <div class="container">
                  <ul class="demo-list-item mdl-list">
        <li class="mdl-list__item">
          <span class="mdl-list__item-primary-content">
            Welcome to the chocolates website.
          </span>
        </li>
        <li class="mdl-list__item">
          <span class="mdl-list__item-primary-content">
            Please see the chocolates you can eat and login to keep track of those
↪you have already eaten!
          </span>
        </li>
      </ul>
    </div>

  </body>

</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="AWS Serverless Web App tutorial">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="Mr. Coxall">
    <meta name="date" content="Jan 2020">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--Google's MDL-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
↪family=Material+Icons">
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.brown-indigo.
↪min.css" />
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>

    <title>Chocolates</title>
  </head>

  <body>
    <!-- Always shows a header, even in smaller screens. -->
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
      <header class="mdl-layout__header">
        <div class="mdl-layout__header-row">
          <!-- Title -->
          <span class="mdl-layout-title">Chocolates</span>
          <!-- Add spacer, to align navigation to the right -->
          <div class="mdl-layout-spacer"></div>
          <!-- Navigation. We hide it in small screens. -->
          <nav class="mdl-navigation mdl-layout--large-screen-only">
            <a class="mdl-navigation__link" href="./index.html">Home</a>
            <a class="mdl-navigation__link" href="./products.html">Product</a>
            <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
            <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
            <a class="mdl-navigation__link" href="./about.html">About</a>
          </nav>
        </div>
      </header>
      <div class="mdl-layout__drawer">
        <span class="mdl-layout-title">Chocolates</span>
        <nav class="mdl-navigation">
          <a class="mdl-navigation__link" href="./index.html">Home</a>
          <a class="mdl-navigation__link" href="./products.html">Product</a>
          <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
          <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
          <a class="mdl-navigation__link" href="./about.html">About</a>
        </nav>
      </div>
      <main class="mdl-layout__content">
        <div class="page-content"><!-- Your content goes here --></div>
      </main>
    </div>

    <!--Get below the nav menu at the top-->
    <br><br><br>

```

(continues on next page)

(continued from previous page)

```
                <div class="container">
      <div>
        <h2>Products</h2>
        <div id='products'>
          <p>...</p>
        </div>
      </div>
                </div>

  </body>

</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="AWS Serverless Web App tutorial">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="Mr. Coxall">
    <meta name="date" content="Jan 2020">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--Google's MDL-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
↪family=Material+Icons">
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.brown-indigo.
↪min.css" />
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>

    <!--Cognito & JavaScript-->
          <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
          <script src="js/amazon-cognito-auth.min.js"></script>
          <script src="https://sdk.amazonaws.com/js/aws-sdk-2.596.0.min.js"></script>
          <script src="./js/amazon-cognito-identity.min.js"></script>
          <script src="./js/config.js"></script>
          <script type="text/javascript" src="./js/sign-in.js"></script>

    <title>Chocolates</title>
  </head>

  <body>
    <!-- Always shows a header, even in smaller screens. -->
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
      <header class="mdl-layout__header">
        <div class="mdl-layout__header-row">
          <!-- Title -->
          <span class="mdl-layout-title">Chocolates</span>
          <!-- Add spacer, to align navigation to the right -->
          <div class="mdl-layout-spacer"></div>
          <!-- Navigation. We hide it in small screens. -->
          <nav class="mdl-navigation mdl-layout--large-screen-only">
            <a class="mdl-navigation__link" href="./index.html">Home</a>
            <a class="mdl-navigation__link" href="./products.html">Product</a>
            <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
            <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
```

(continues on next page)

(continued from previous page)

```
            <a class="mdl-navigation__link" href="./about.html">About</a>
          </nav>
        </div>
      </header>
      <div class="mdl-layout__drawer">
        <span class="mdl-layout-title">Chocolates</span>
        <nav class="mdl-navigation">
          <a class="mdl-navigation__link" href="./index.html">Home</a>
          <a class="mdl-navigation__link" href="./products.html">Product</a>
          <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
          <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
          <a class="mdl-navigation__link" href="./about.html">About</a>
        </nav>
      </div>
      <main class="mdl-layout__content">
        <div class="page-content"><!-- Your content goes here --></div>
      </main>
    </div>

    <!--Get below the nav menu at the top-->
    <br><br><br>

                <div class="container">
      <div>
        <h2>Sign In</h2>
        <div id='sign-in'>
        </div>
      </div>
                </div>

    <div class="mdl-layout mdl-js-layout mdl-color--grey-100">
            <main class="mdl-layout__content">
                    <div class="mdl-card mdl-shadow--6dp">
                            <div class="mdl-card__title mdl-color--primary mdl-color-
↪text--white">
                                    <h2 class="mdl-card__title-text">Chocolate Login</
↪h2>
                            </div>
                      <div class="mdl-card__supporting-text">
                                    <form action="#">
                                            <div class="mdl-textfield mdl-js-textfield
↪">
                                                    <input class="mdl-textfield__input
↪" type="text" id="inputUsername" />
                                                    <label class="mdl-textfield__label
↪" for="username">Email Address</label>
                                            </div>
                                            <div class="mdl-textfield mdl-js-textfield
↪">
                                                    <input class="mdl-textfield__input
↪" type="password" id="inputPassword" />
                                                    <label class="mdl-textfield__label
↪" for="userpass">Password</label>
                                            </div>
                                    </form>
                            </div>
                            <div class="mdl-card__actions mdl-card--border">
```

(continues on next page)

(continued from previous page)

```
                                    <button class="mdl-button mdl-button--colored mdl-
↪js-button mdl-js-ripple-effect" onclick="signInButton();">Log in</button>
                            </div>
                    </div>
            </main>
    </div>

    <div class="container">
      <div>
        <div id='logged-in'>
          <p></p>
        </div>
      </div>
                </div>

  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="AWS Serverless Web App tutorial">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="Mr. Coxall">
    <meta name="date" content="Jan 2020">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--Google's MDL-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
↪family=Material+Icons">
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.brown-indigo.
↪min.css" />
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>

    <!--Cognito & JavaScript-->
                <script src="./js/amazon-cognito-identity.min.js"></script>
                <script src="./js/config.js"></script>
                <script src="./js/sign-out.js"></script>

    <title>Chocolates</title>
  </head>

  <body>
    <!-- Always shows a header, even in smaller screens. -->
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
      <header class="mdl-layout__header">
        <div class="mdl-layout__header-row">
          <!-- Title -->
          <span class="mdl-layout-title">Chocolates</span>
          <!-- Add spacer, to align navigation to the right -->
          <div class="mdl-layout-spacer"></div>
          <!-- Navigation. We hide it in small screens. -->
          <nav class="mdl-navigation mdl-layout--large-screen-only">
            <a class="mdl-navigation__link" href="./index.html">Home</a>
            <a class="mdl-navigation__link" href="./products.html">Product</a>
```

(continues on next page)

(continued from previous page)

```
            <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
            <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
            <a class="mdl-navigation__link" href="./about.html">About</a>
          </nav>
        </div>
      </header>
      <div class="mdl-layout__drawer">
        <span class="mdl-layout-title">Chocolates</span>
        <nav class="mdl-navigation">
          <a class="mdl-navigation__link" href="./index.html">Home</a>
          <a class="mdl-navigation__link" href="./products.html">Product</a>
          <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
          <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
          <a class="mdl-navigation__link" href="./about.html">About</a>
        </nav>
      </div>
      <main class="mdl-layout__content">
        <div class="page-content"><!-- Your content goes here --></div>
      </main>
    </div>

    <!--Get below the nav menu at the top-->
    <br><br><br>

                <div class="container">
      <div>
        <h2>Sign Out</h2>
        <div id='sign-out'>
          <p>One moment please ...</p>
        </div>
      </div>
                </div>

    <br>
  </body>
        <script>
                window.onload = function(){
      const temp_var = signOut();
                }
  </script>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="AWS Serverless Web App tutorial">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="Mr. Coxall">
    <meta name="date" content="Jan 2020">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--Google's MDL-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
↪family=Material+Icons">
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.brown-indigo.
↪min.css" />
```

(continues on next page)

(continued from previous page)

```
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>

    <title>Chocolates</title>
  </head>

  <body>
    <!-- Always shows a header, even in smaller screens. -->
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
      <header class="mdl-layout__header">
        <div class="mdl-layout__header-row">
          <!-- Title -->
          <span class="mdl-layout-title">Chocolates</span>
          <!-- Add spacer, to align navigation to the right -->
          <div class="mdl-layout-spacer"></div>
          <!-- Navigation. We hide it in small screens. -->
          <nav class="mdl-navigation mdl-layout--large-screen-only">
            <a class="mdl-navigation__link" href="./index.html">Home</a>
            <a class="mdl-navigation__link" href="./products.html">Product</a>
            <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
            <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
            <a class="mdl-navigation__link" href="./about.html">About</a>
          </nav>
        </div>
      </header>
      <div class="mdl-layout__drawer">
        <span class="mdl-layout-title">Chocolates</span>
        <nav class="mdl-navigation">
          <a class="mdl-navigation__link" href="./index.html">Home</a>
          <a class="mdl-navigation__link" href="./products.html">Product</a>
          <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
          <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
          <a class="mdl-navigation__link" href="./about.html">About</a>
        </nav>
      </div>
      <main class="mdl-layout__content">
        <div class="page-content"><!-- Your content goes here --></div>
      </main>
    </div>

    <!--Get below the nav menu at the top-->
    <br><br><br>

                <div class="container">
      <div>
        <h2>About</h2>
        <div id='about'>
          <p>...</p>
        </div>
      </div>
                </div>

  </body>

</html>
```

```
<!DOCTYPE html>
```

(continues on next page)

(continued from previous page)

```
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="AWS Serverless Web App tutorial">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="Mr. Coxall">
    <meta name="date" content="Jan 2020">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--Google's MDL-->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.brown-indigo.min.css" />
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>

    <!--Cognito & JavaScript-->
    <script src="js/amazon-cognito-identity.min.js"></script>
    <script src="js/config.js"></script>
    <script type="text/javascript" src="./js/profile.js"></script>

    <title>Chocolates</title>
  </head>

  <body>
    <!-- Always shows a header, even in smaller screens. -->
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
      <header class="mdl-layout__header">
        <div class="mdl-layout__header-row">
          <!-- Title -->
          <span class="mdl-layout-title">Chocolates</span>
          <!-- Add spacer, to align navigation to the right -->
          <div class="mdl-layout-spacer"></div>
          <!-- Navigation. We hide it in small screens. -->
          <nav class="mdl-navigation mdl-layout--large-screen-only">
            <a class="mdl-navigation__link" href="./index.html">Home</a>
            <a class="mdl-navigation__link" href="./products.html">Product</a>
            <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
            <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
            <a class="mdl-navigation__link" href="./about.html">About</a>
          </nav>
        </div>
      </header>
      <div class="mdl-layout__drawer">
        <span class="mdl-layout-title">Chocolates</span>
        <nav class="mdl-navigation">
          <a class="mdl-navigation__link" href="./index.html">Home</a>
          <a class="mdl-navigation__link" href="./products.html">Product</a>
          <a class="mdl-navigation__link" href="./sign-in.html">Sign In</a>
          <a class="mdl-navigation__link" href="./sign-out.html">Sign Out</a>
          <a class="mdl-navigation__link" href="./about.html">About</a>
        </nav>
      </div>
      <main class="mdl-layout__content">
        <div class="page-content"><!-- Your content goes here --></div>
      </main>
    </div>
```

(continues on next page)

(continued from previous page)

```
    <!--Get below the nav menu at the top-->
    <br><br><br>

                <div class="container">
      <div>
        <h2>Profile</h2>
        <div id='profile'>
        </div>
      </div>
                </div>

                <div class="container">
      <table class="mdl-data-table mdl-js-data-table mdl-data-table--selectable">
        <thead>
          <tr>
            <th class="mdl-data-table__cell--non-numeric">Email</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td class="mdl-data-table__cell--non-numeric">qwerty@qwerty.com</td>
            <td>Qwerty</td>
            <td>Qwerty</td>
            <td>15</td>
          </tr>
          <tr>
            <td class="mdl-data-table__cell--non-numeric"><div id='profile_email'></
↪div></td>
            <td><div id='profile_first_name'></div></td>
            <td><div id='profile_last_name'></div></td>
            <td><div id='profile_age'></div></td>
          </tr>
        </tbody>
      </table>
                </div>

  </body>

  <script>
                window.onload = function(){
      const temp_var = getUserAttributes();
                }
  </script>

</html>
```

15.2 JavaScript

Home

Products

Sign In

Sign Out

About

Profile

```
// no javascript code
```

```
// no javascript code
```

```
// JavaScript File

function signInButton() {
  // sign-in to AWS Cognito

  var data = {
          UserPoolId : _config.cognito.userPoolId,
    ClientId : _config.cognito.clientId
  };
  var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
  var cognitoUser = userPool.getCurrentUser();

        var authenticationData = {
    Username : document.getElementById("inputUsername").value,
    Password : document.getElementById("inputPassword").value,
  };

  var authenticationDetails = new AmazonCognitoIdentity.
↪AuthenticationDetails(authenticationData);

  var poolData = {
    UserPoolId : _config.cognito.userPoolId, // Your user pool id here
    ClientId : _config.cognito.clientId, // Your client id here
  };

  var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);

  var userData = {
    Username : document.getElementById("inputUsername").value,
    Pool : userPool,
  };

  var cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);

  cognitoUser.authenticateUser(authenticationDetails, {
    onSuccess: function (result) {
      var accessToken = result.getAccessToken().getJwtToken();
      console.log(result);

      //get user info, to show that you are logged in
                        cognitoUser.getUserAttributes(function(err, result) {
                                if (err) {
                                        console.log(err);
                                        return;
                                }
                                console.log(result);
```

(continues on next page)

(continued from previous page)

```
                                document.getElementById("logged-in").innerHTML = "You
↪are logged in as: " + result[2].getValue();

                                // now auto redirect to profile page
                                window.location.replace("./profile.html");
                            });

    },
    onFailure: function(err) {
      alert(err.message || JSON.stringify(err));
    },
  });
}
```

```
// JavaScript File

function signOut() {
  //

  return_message = "";

  const data = {
          UserPoolId : _config.cognito.userPoolId,
    ClientId : _config.cognito.clientId
  };
  const userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
  const cognitoUser = userPool.getCurrentUser();

        if (cognitoUser != null) {
          cognitoUser.getSession(function(err, session) {
      if (err) {
              alert(err);
        return;
      }
      console.log('session validity: ' + session.isValid());

                        // sign out
                        cognitoUser.signOut();
                        console.log("Signed-out");
                        return_message = "Signed-out";
          });
        } else {
                console.log("Already signed-out")
                return_message = "Already signed-out";
        }

        const div_user_info = document.getElementById('sign-out');
  div_user_info.innerHTML = return_message;
}
```

```
// no javascript code
```

```
// JavaScript File

async function getUser(email_address) {
```

(continues on next page)

(continued from previous page)

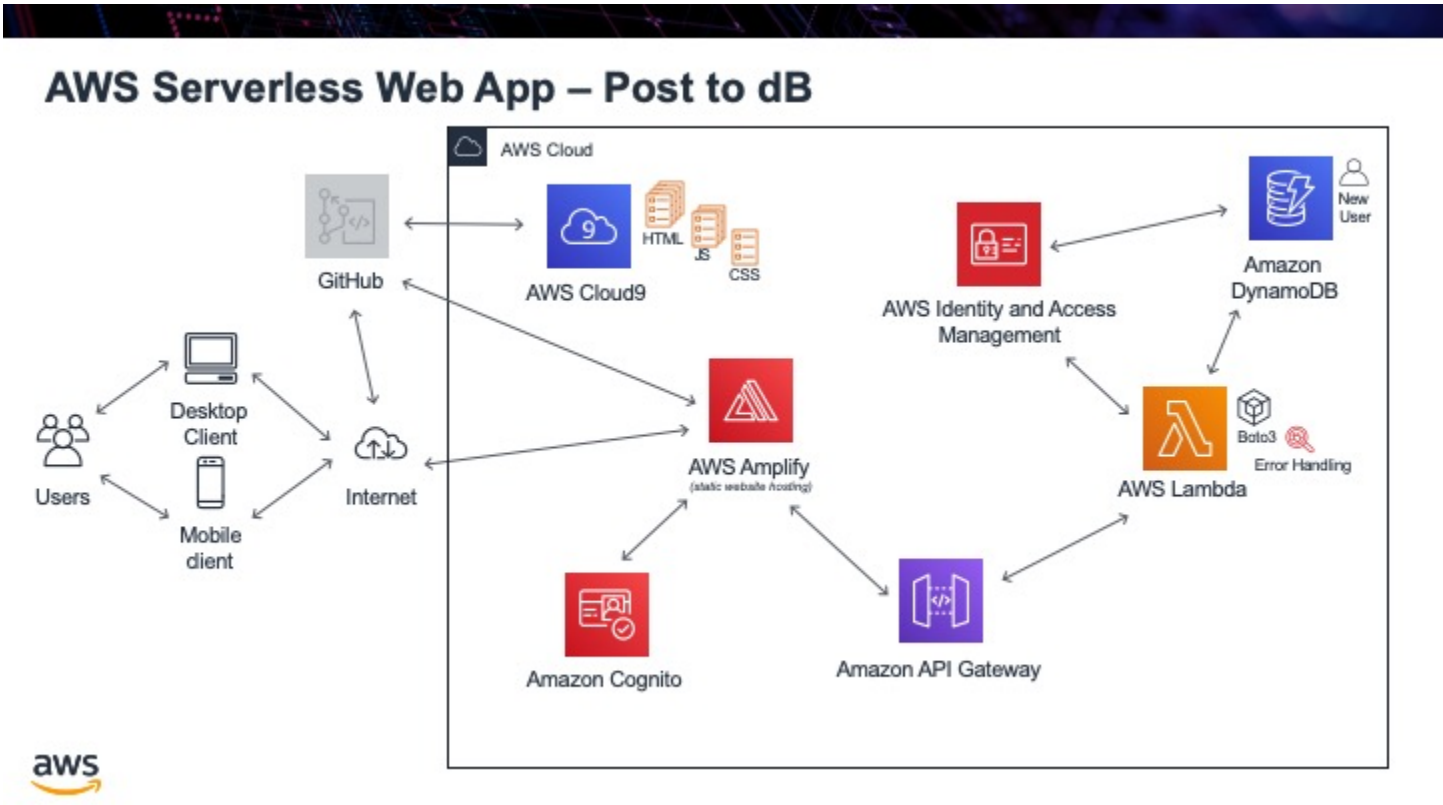
```
    // get the user info from API Gate

    const api_url = 'https://gonvpjbyuf.execute-api.us-east-1.amazonaws.com/prod/user-profile?user_email=' + email_address;
    const api_response = await fetch(api_url);
    const api_data = await(api_response).json();
    console.log(api_data);

    const json_profile = JSON.parse(api_data['body']);
    const div_user_profile_email = document.getElementById('profile_email');
    const div_user_profile_first_name = document.getElementById('profile_first_name');
    const div_user_profile_last_name = document.getElementById('profile_last_name');
    const div_user_profile_age = document.getElementById('profile_age');

    div_user_profile_email.innerHTML = json_profile['email'];
    div_user_profile_first_name.innerHTML = json_profile['first_name'];
    div_user_profile_last_name.innerHTML = json_profile['last_name'];
    div_user_profile_age.innerHTML = json_profile['age'];
  }

function getUserAttributes() {
        var data = {
                UserPoolId : _config.cognito.userPoolId,
    ClientId : _config.cognito.clientId
        };
        var userPool = new AmazonCognitoIdentity.CognitoUserPool(data);
        var cognitoUser = userPool.getCurrentUser();

        if (cognitoUser != null) {
          cognitoUser.getSession(function(err, session) {
      if (err) {
              alert(err);
        return;
      }
      //console.log('session validity: ' + session.isValid());

      cognitoUser.getUserAttributes(function(err, result) {
                                if (err) {
                                        console.log(err);
                                        return;
                                }
                                // user email address
                                console.log(result[2].getValue());
                                getUser(result[2].getValue())
                        });

          });
        } else {
                console.log("Already signed-out")
        }
}
```

See also:

Google's Material Design Lite website

CHAPTER 16

Post to dataBase



The next step we need to be able to do is create a new row in the database. To do this we run the command **put_Item**. We will once again have to make a Lambda function to run this code. We will not actually create an API Gateway to access this code, instead in the next step we will get Cognito to automatically **trigger** this Lambda function when a new user is created.

The reason we need this is that Cognito is taking care of the user sign-in for us. But when a new user registeres Cognito knows about the new user, but our DynamoDB does not. What we need to hap-

pen is for Cognito to add a new row with the users email address into our database. Since Cognito is sending the information to Lambda, once again as a JSON file, we will need to know what this JSON file will look like, so we can parse it appropriately. From this https://stackoverflow.com/questions/49580078/send-username-to-aws-lambda-function-triggered-by-aws-cognito-user-confirm website we know that the JSON file Cognito will send is like this:

Listing 1: Lambda function parameters from Cognito for post-confirmation trigger

```
{
  "version": "1",
  "region": "eu-central-1",
  "userPoolId": "eu-central-1_45YtlkflA",
  "userName": "user4",
  "callerContext": {
      "awsSdkVersion": "aws-sdk-java-console",
      "clientId": "4736lckau64in48dku3rta0eqa"
  },
  "triggerSource": "PostConfirmation_ConfirmSignUp",
  "request": {
      "userAttributes": {
          "sub": "a2c21839-f9fc-49e3-be9a-16f5823d6705",
          "cognito:user_status": "CONFIRMED",
          "email_verified": "true",
          "email": "asdfsdfsgdfg@carbtc.net"
      }
  },
  "response": {}
}
```

Therefore, since we are looking for the email address of the user the very bearried piece of information we are looking for is: event['request']['userAttributes']['email']. We will also only be entering in the email address column inot the database. THis is ok, since it is the primary key, and the rest of the information the user can add in a **Profile Edit** page. It also turns out that Cognito is very particular about what it gets back as a response. Before we were sending back JSON files for the web, but this is not going to the web. Cognito is actually looking for the **event** object to be retruned to it. This signals that everything went OK. Anything else is considered an error.

Tasks:

- create Lambda function called **add_user**
- write code to create the row in our database
- write test case to see new row show up in database, ensuring we are using the sample JSON file Conginto will send us
- test it out and seeing row show up

Listing 2: Lambda function to create row in DynamoDB

```
#!/usr/bin/env python3

# Created by: Mr. Coxall
# Created on: Dec 2019
# This function adds a row from our chocolate_user DynmamoDB

import json
import boto3

```

(continues on next page)

(continued from previous page)

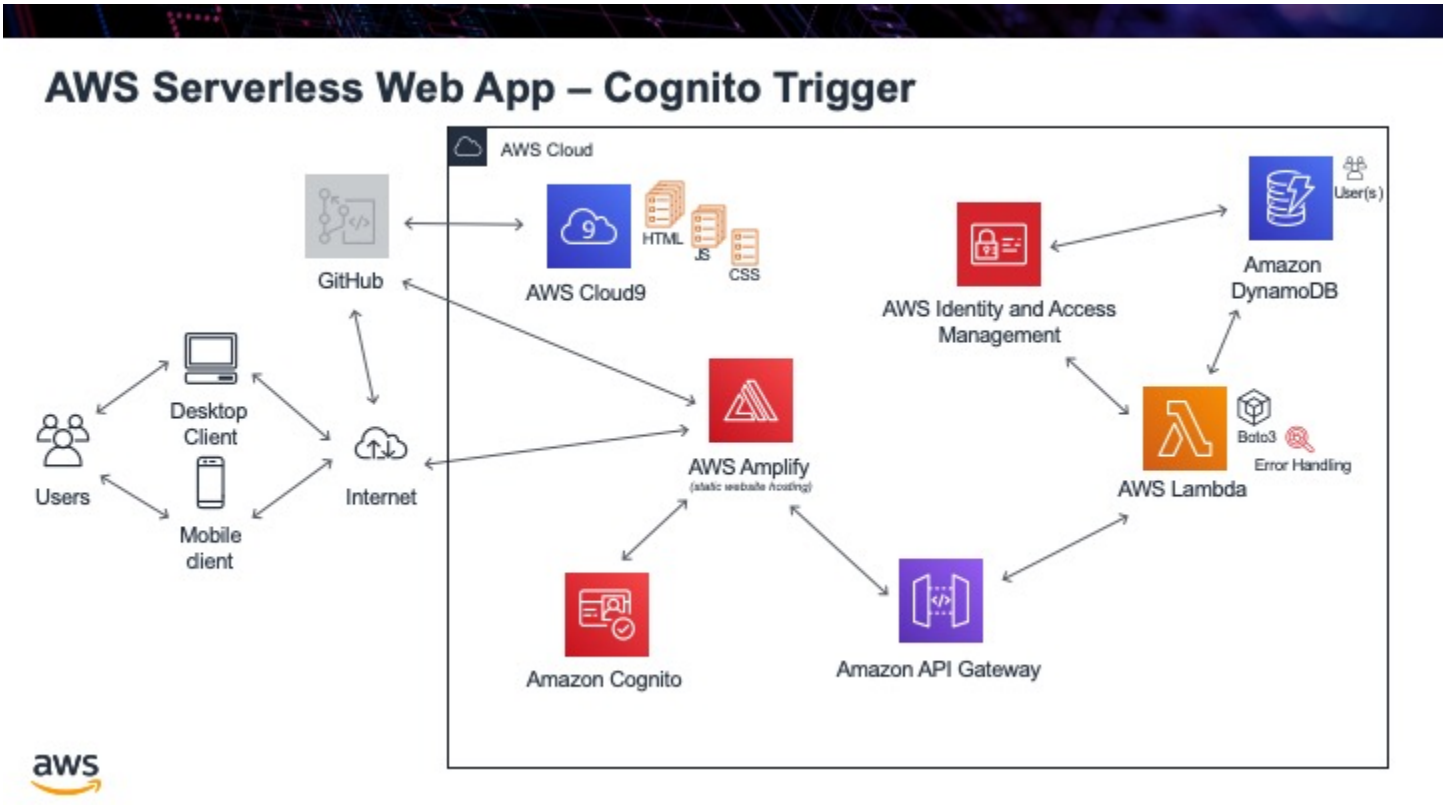
```
10
11  def lambda_handler(event, context):
12      # function returns a row from our chocolate_user DynmamoDB
13
14      dynamodb = boto3.resource('dynamodb')
15      table = dynamodb.Table('chocolate_user')
16
17      try:
18          response = table.put_item(
19              Item = {
20                  'email': event['request']['userAttributes']['email']
21              }
22          )
23
24          try:
25              result = response['ResponseMetadata']
26          except:
27              result = {}
28
29          print(result)
30          return_var = json.dumps(result)
31
32          print(result)
33
34          # Cognito is expecting the "event" object to be returned for success
35          return event
36
37      except:
38         return "error"
```

See also:

StackOVerflow Send username to aws Lambda function triggered by aws Cognito user confirm

CHAPTER 17

Cognito Trigger



As mentioned in the last step, the reason to create a Lambda function to add a row to our **User** table, is for when a new user registers, we need a way to not only have Cognito know the user but also our DynamoDB as well. We now have a Lambda function that will create a new row. Now we need Cognito to trigger is.

If you go back to the user pool in Cognito that you created previously and then select **Triggers** on the left hand side, you will see a list of triggers that Cognito can execute during different stages of its process. The one we are interested in is **Post confirmation**. This occurs after our user has confirmed themeself by clicking on the link in the email that is

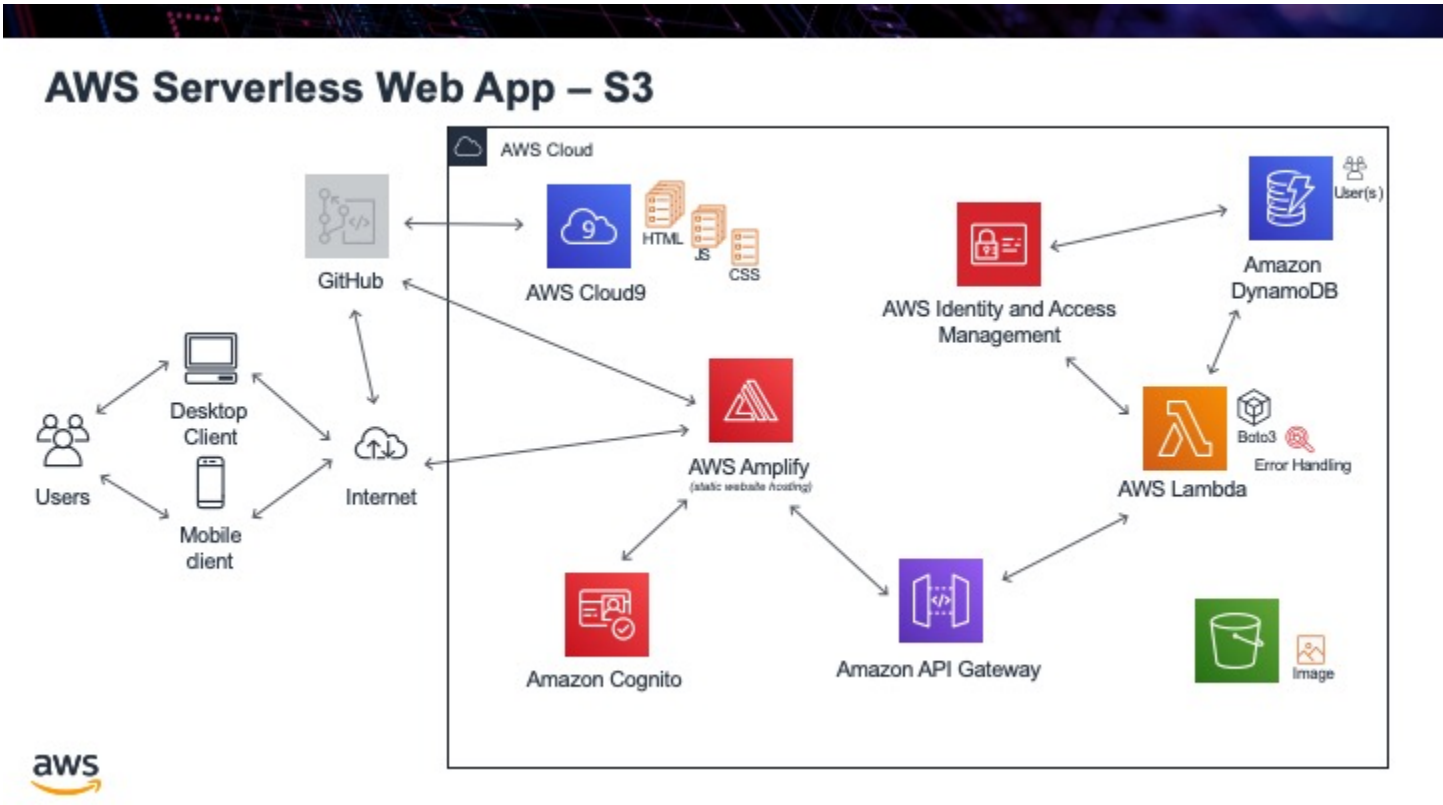
sent to them. Once we have confirmed the user, we will trigger our Lambda function. To do this just select it from the dropdown menu and then **Save changes** at the bottom.

Tasks:

- link our Lambda function to Cognito trigger
- test it out by creating a new user
- confirm the new user exists in the DynamoDB table

CHAPTER 18

S3



A next nice step would be if we could upload an image to use as the profile image of the user. The problem is that an image cannot be saved into our DynamoDB table. It can only hold text or numbers. What we can do instead is create an S3 bucket and save our images into it. Then we just save the **key** in our table, that points to our particular image.

Before we do anything else, if we want to access the image later with a Lambda function to return the image to the web, the Lambda function need permission to access S3. Go back to IAm and add the permission **AmazonS3FullAccess** to our "Role". Once this is done, go to S3 and create a bucket. By defauly every bucket on S3 for all users must have a

unique name, so you will not be able to use the name I gave it, but just remember your name. Kepp all the defaults as is.

Next we have to enable **CORS** again, so that different domains can be used in our web application. Here is the CORS JSON file permissions:

Listing 1: CORS permissions for S3

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
</CORSRule>
</CORSConfiguration>
```

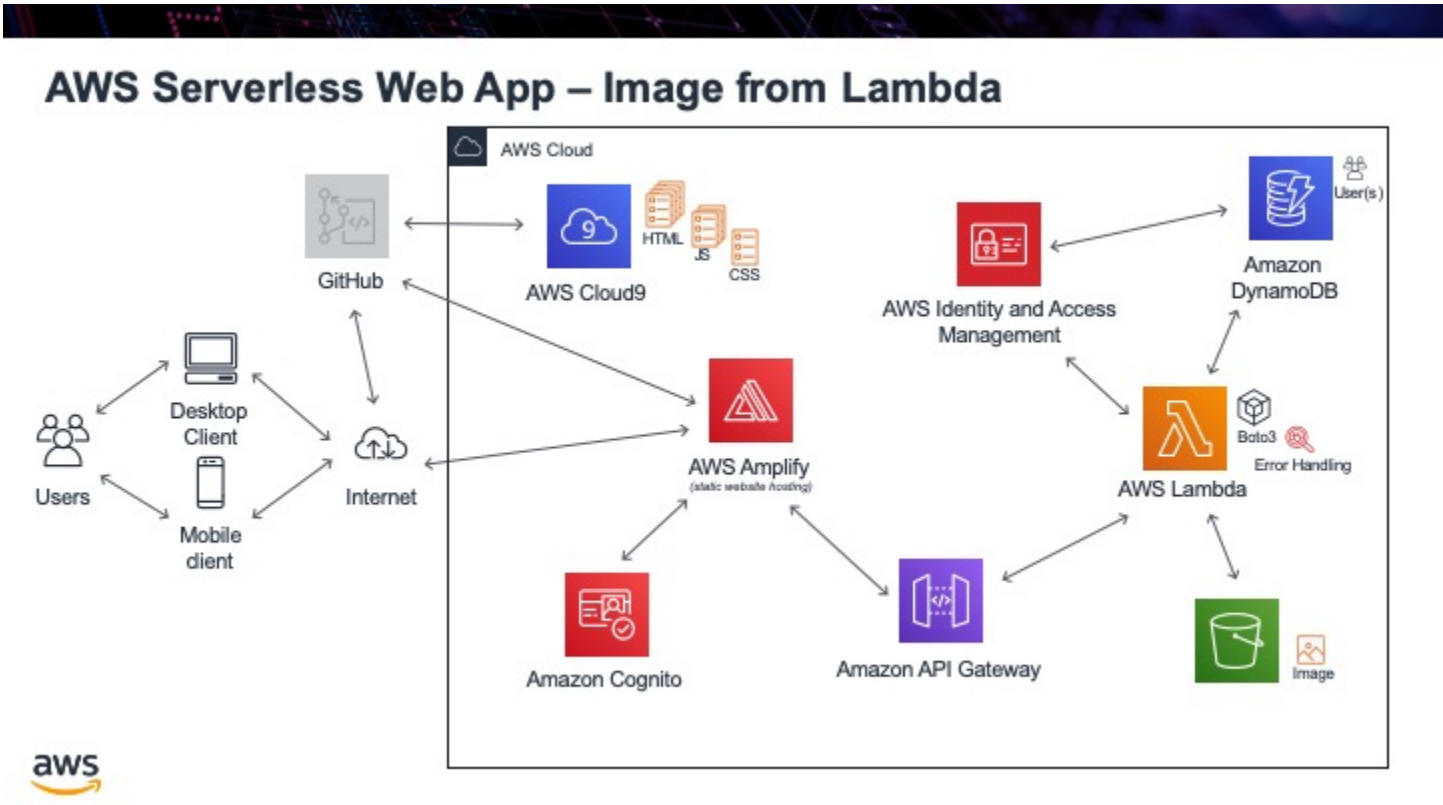
Upload and image to the S3 bucket. For consistancy we will use the email address of the user as the file name. For example: mr.coxall@mths.ca.jpg will be the file name. I know using the "@" is highly unusual, but it will work.

Tasks:

- add AmazonS3FullAccess to our IAM Roll
- create S3 bucket
- upload an image to the bucket, using the email address as the file name

CHAPTER 19

Get Image from Lambda



Now that we have our image in S3, we need to be able to get it out using Lambda. To do that create a new Lambda function called **get_profile_image**. One problem we have is that we have to pass the image eventually to API Gateway as a JSON file. We can not pass it as a ".jpg" file. To do this we will access the image, convert it to **base64** and then pass it out of our function as a JSON file. Our JavaScript function on our web front end will then un-encode it and present it back as an image.

Here is the Lambda function code:

Listing 1: Get S3 image and return it as JSON

```
import boto3
import base64
from boto3 import client

def lambda_handler(event, context):
    # got this function from: https://stackoverflow.com/questions/36240356/lambda-get-image-from-s3
    profile_image_key = event['email_address']
    user_download_img = profile_image_key + '.jpg'

    s3 = boto3.resource('s3')
    bucket = s3.Bucket(u'coxall-profile-photo')
    obj = bucket.Object(key = user_download_img)      #pass your image Name to key
    response = obj.get()     #get Response
    img = response[u'Body'].read()        # Read the respone, you can also print it.
    #print(type(img))                      # Just getting type.
    myObj = [base64.b64encode(img)]          # Encoded the image to base64
    #print(type(myObj))             # Printing the values
    #print(myObj[0])                # get the base64 format of the image
    #print('type(myObj[0]) ================>',type(myObj[0]))
    return_json = str(myObj[0])           # Assing to return_json variable to return.
    #print('return_json ======================>',return_json)
    return_json = return_json.replace("b'","")          # replace this 'b'' is must to get absoulate image.
    encoded_image = return_json.replace("'","")

    return {
        'status': 'True',
        'statusCode': 200,
        'message': 'Downloaded profile image',
        'encoded_image':encoded_image          # returning base64 of your image which in s3 bucket.
    }
# to prove that this is returning the image goto this website
# https://codebeautify.org/base64-to-image-converter
# place the "encoded_image" into "Enter Base64 String"
# when doing so, ensure you do not include the quotes
# you should get back your image
# :)
```

Use this URL (https://codebeautify.org/base64-to-image-converter) to prove that what you are getting back from the Lambda function really is the image.

Tasks:

- create Lambda function, get_profile_image
- create test case to get back example JSON file
- prove JSON file does contain the image