SERVERLESS WEBAPP HOSTING ON AWS

MAJOR PROJECT

Prepared by:

Viransh Bhardwaj

Yukta Rajesh Sharma

Gaddala Greeshma Devi

Sarvesh Lath

Vikas Reddy

Cloud Computing Batch July 2022

MENTOR: SHRAVANI SHETTY.

Web Application Hosting on AWS

Web Apps have always been required to be hosted on servers and are assigned an IP that allows the users anywhere in the world to access the webapps using that IP. PuTTY allows the integration of the instance IP with the S3 Bucket Objects and launches an Apache Server to host the webapp.

Traditional on-premises web architectures require complex solutions and accurate reserved capacity forecast in order to ensure reliability. Dense peak traffic periods and wild swings in traffic patterns result in low utilization rates of expensive hardware. This yields high operating costs to maintain idle hardware, and an inefficient use of capital for underused hardware.

Amazon Web Services (AWS) provides a reliable, scalable, secure, and highly performing infrastructure for the most demanding web applications. This infrastructure matches IT costs with customer traffic patterns in near-real time.

This whitepaper is meant for IT Managers and System Architects who want to understand how to run traditional web architectures in the cloud to achieve elasticity, scalability, and reliability.

Static website: A static website contains Web pages with fixed content. Each page is coded in HTML and displays the same information to every visitor. Static sites are the most basic type of website and are the easiest to create. Unlike dynamic websites, they do not require any Web programming or database design. A static site can be built by simply creating a few HTML pages and publishing them to a Web server.

Since static Web pages contain fixed code, the content of each page does not change unless it is manually updated by the webmaster. This works well for small websites, but it can make large sites with hundreds or thousands of pages difficult to maintain. Therefore, larger websites typically use dynamic pages, which can be updated by simply modifying a database record. Static sites that contain a lot of pages are often designed using templates. This makes it possible to update several pages at once, and also helps provide a consistent layout throughout the site.

AWS LAMBDA SERVICE

Lambda falls under “Compute” service in AWS (Amazon Web Services). Using Lambda, we can code without provisioning or managing servers. Lambda automatically runs our code without requiring us to provision or manage servers. We just need to write the code and upload it to the Lambda Function. Lambda executes the code only when needed. It grows automatically supporting from a few requests to thousands of requests. We are charged for every 100ms our code executes and the number of times it is triggered. We are charged only for the compute time our code consumes and not charged when the code is not being executed.

Everything to know about Amazon Dynamo DB

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

With DynamoDB, you can create database tables that can store and retrieve any amount of data and serve any level of request traffic. You can scale up or scale down your tables' throughput capacity without downtime or performance degradation. You can use the AWS Management Console to monitor resource utilization and performance metrics. DynamoDB provides on-demand backup capability. It allows you to create full backups of your tables for long-term retention and archival for regulatory compliance needs.

High availability and Durability:-

DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements, while maintaining consistent and fast performance. All of your data is stored on solid-state disks (SSDs) and is automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability. You can use global tables to keep DynamoDB tables in sync across AWS Regions.

Core components of Amazon Dynamo DB:-

In DynamoDB, tables, items, and attributes are the core components that you work with. A *table* is a collection of *items*, and each item is a collection of *attributes*. DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility. You can use DynamoDB Streams to capture data modification events in DynamoDB tables.

Basic components of Dynamo DB:-

Tables–Similar to other data base systems, Dynamo DB stores data in tables. A *table* is a collection of data. For example, see the example table called *People* that you could use to store personal contact information about friends, family, or anyone else of interest. You could also have a *Cars* table to store information about vehicles that people drive.

Items–
Eachtablecontainszeroormoreitems.An*item*isagroupofattributesthatisuniquely identifiable among all of the other items. In a *People* table, each item represents a person. For a *Cars* table, each item represents one vehicle. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. In DynamoDB, there is no limit to the number of items you can store in a table.

Attributes – Each item is composed of one or more attributes. An *attribute* is a fundamental data element, something that does not need to be broken

down any further. For example, an item in a *People* table contains attributes called *PersonID*, *LastName*, *FirstName*, and so on. For a *Department* table, an item might have attributes such as *DepartmentID*, *Name*, *Manager*, and so on. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

Dynamo DB streams: -

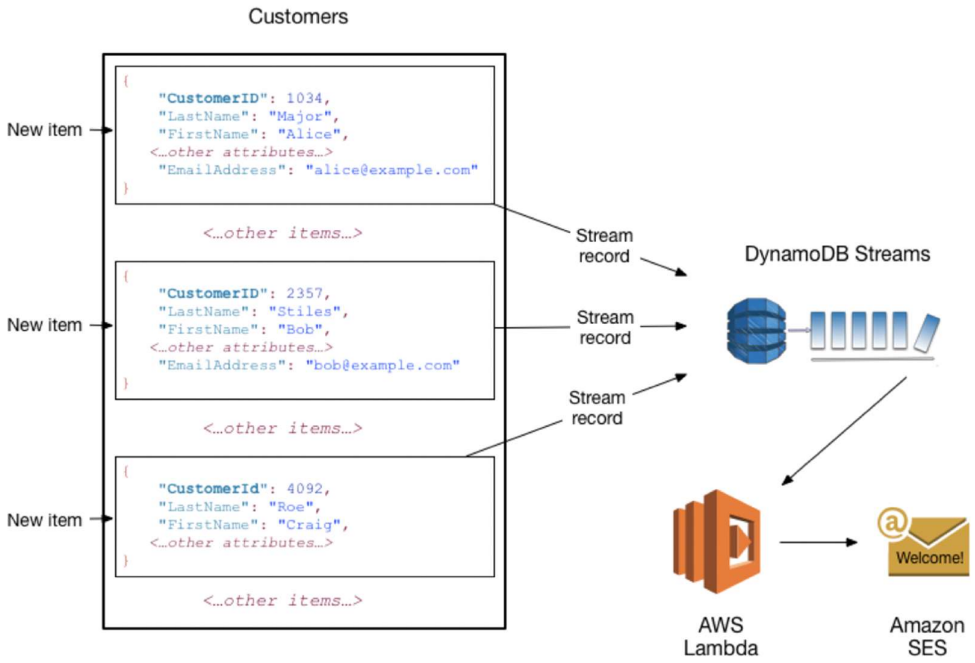
DynamoDB Streams is an optional feature that captures data modification events in DynamoDB tables. The data about these events appear in the stream in near-real time, and in the order that the events occurred.

Each event is represented by a *stream record*. If you enable a stream on a table, DynamoDB Streams writes a stream record whenever one of the following events occurs:

- A new item is added to the table: The stream captures an image of the entire item, including all of its attributes.
- An item is updated: The stream captures the "before" and" after " image of any attributes that were modified in the item.
- An item is deleted from the table: The stream captures an image of the entire item before it was deleted.

  Each stream record also contains the name of the table, the event timestamp, and other metadata. Stream records have a lifetime of 24 hours; after that, they are automatically removed from the stream.

  You can use DynamoDB Streams together with AWS Lambda to create a *trigger*—code that runs automatically whenever an event of interest appears in a stream. For example, consider a *Customers* table that contains customer information for a company. Suppose that you want to send a "welcome" email to each new customer. You could enable a stream on that table, and then associate the stream with a Lambda function. The Lambda function would run whenever a new stream record appears, but only process new items added to the *Customers* table. For any item that has an `Email Address` attribute, the Lambda function would invoke Amazon Simple Email Service (Amazon SES) to send an email to that address.



Dynamo DB API:-

To work with Amazon DynamoDB, your application must use a few simple API operations. The following is a summary of these operations, organized by category.

Topics

- **Controlplane**
- **Data plane**
- **DynamoDBStreams**
- **Transactions**

What is Amazon API Gateway?

Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. As an API Gateway API developer, you can create APIs for use in your own client applications. Or you can make your APIs available to third-party app developers. For more information, see the section called “Who uses API Gateway?”

API Gateway creates RESTful APIs that:

• Are HTTP-based.

• Enable stateless client-server communication.

• Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

For more information about API Gateway REST APIs and HTTP APIs, see the section called “Choosing between REST APIs and HTTP APIs ” , Working with HTTP APIs , the section called “Use API Gateway to create REST APIs” , and the section called “Create and configure”

API Gateway creates WebSocket APIs that:

• Adhere to the WebSocket protocol, which enables stateful, full-duplex communication between client and server.

• Route incoming messages based on message content.

For more information about API Gateway WebSocket APIs, see the section called “Use API Gateway to create WebSocket APIs” and the section called “About WebSocket APIs” .

Topics

• Architecture of API Gateway

• Features of API Gateway

• API Gateway use cases

• Accessing API Gateway

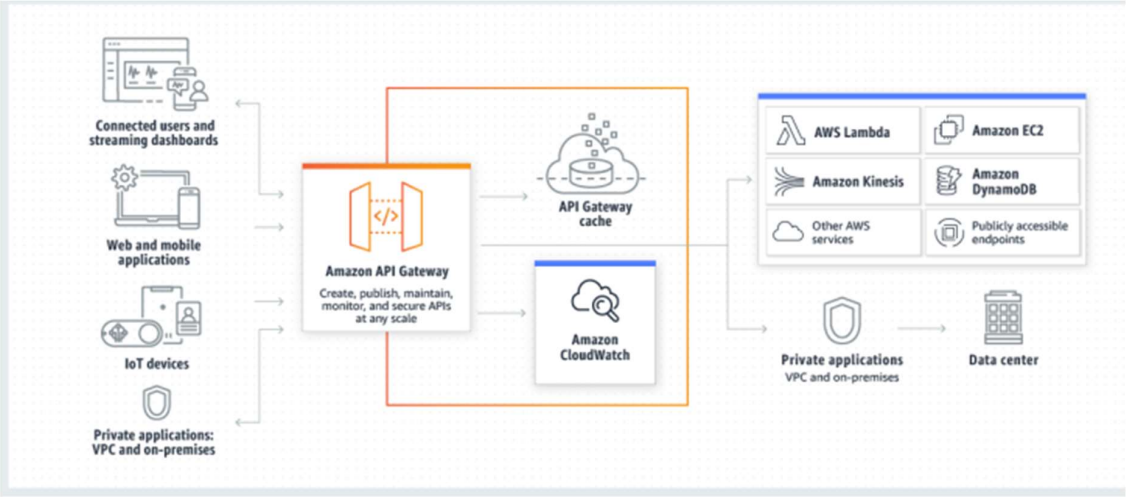
• Part of AWS serverless infrastructure

• How to get started with Amazon API Gateway

• Amazon API Gateway concepts

Architecture of API Gateway

The following diagram shows API Gateway architecture.



This diagram illustrates how the APIs you build in Amazon API Gateway provide you or your developer customers with an integrated and consistent developer experience for building AWS serverless applications. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls. These tasks include traffic management, authorization and access control, monitoring, and API version management.

API Gateway acts as a "front door" for applications to access data, business logic, or functionality from your backend services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, any web application, or real-time communication applications.

Features of API Gateway:

• Powerful, flexible authentication mechanisms, such as AWS Identity and Access Management policies, Lambda authorizer functions, and Amazon Cognito user pools.

CloudWatch access logging and execution logging, including the ability to set alarms. For more information, see the section called “CloudWatch metrics” (p. 618) and the section called “Metrics”.

• Ability to use AWS CloudFormation templates to enable API creation. For more information, see Amazon API Gateway Resource Types Reference and Amazon API Gateway V2 Resource Types Reference.

API Gateway use cases:

Topics

- Use API Gateway to create REST APIs
- Use API Gateway to create HTTP APIs
- Use API Gateway to create WebSocket APIs
- Who uses API Gateway?

An API Gateway REST API is made up of resources and methods. A resource is a logical entity that an app can access through a resource path. A method corresponds to a REST API request that is submitted by the user of your API and the response returned to the user. For example, /incomes could be the path of a resource representing the income of the app user. A resource can have one or more operations that are defined by appropriate HTTP verbs such as GET, POST, PUT, PATCH, and DELETE. A combination of a resource path and an operation identifies a method of the API. For example, a POST /incomes method could add an income earned by the caller, and a GET / expenses method could query the reported expenses incurred by the caller. The app doesn't need to know where the requested data is stored and fetched from on the backend. In API Gateway REST APIs, the frontend is encapsulated by method requests and method responses. The API interfaces with the backend by means of integration requests and integration responses. For example, with DynamoDB as the backend, the API developer sets up the integration request to forward the incoming method request to the chosen backend. The setup includes specifications of an appropriate DynamoDB action, required IAM role and policies, and required input data transformation. The backend returns the result to API Gateway as an integration response.

HTTP APIs enable you to create RESTful APIs with lower latency and lower cost than REST APIs. You can use HTTP APIs to send requests to AWS Lambda functions or to any publicly routable HTTP endpoint. For example, you can create an HTTP API that integrates with a Lambda function on the backend. When a client calls your API, API Gateway sends the request to the Lambda function and returns the function's response to the client. HTTP APIs support OpenID Connect and OAuth 2.0 authorization. They come with built-in support for cross-origin resource sharing (CORS) and automatic deployments.

In a WebSocket API, the client and the server can both send messages to each other at any time. Backend servers can easily push data to connected users and devices, avoiding the need to implement complex polling mechanisms. For example, you could build a serverless application using an API Gateway WebSocket API and AWS Lambda to send and receive messages to and from individual users or groups of users in a chat room. Or you could invoke backend services such as AWS Lambda, Amazon Kinesis, or an HTTP endpoint based on message content. You can use API Gateway WebSocket APIs to build secure, real-time communication applications without having to provision or manage any servers to manage connections or large-scale data exchanges.

Targeted use cases include real-time applications such as the following:

- Chat applications • Real-time dashboards such as stock tickers • Real-time alerts and notifications

Who uses API Gateway?

There are two kinds of developers who use API Gateway: API developers and app developers. An API developer creates and deploys an API to enable the required functionality in API Gateway. The API developer must be an IAM user in the AWS account that owns the API. An app developer builds a functioning application to call AWS services by invoking a WebSocket or REST API created by an API developer in API Gateway. The app developer is the customer of the API developer. The app developer doesn't need to have an AWS account, provided that the API either doesn't require IAM permissions or supports authorization of users through third-party federated identity providers supported by Amazon Cognito user pool identity federation. Such identity providers include Amazon, Amazon Cognito user pools, Facebook, and Google.

Accessing API Gateway

• AWS Management Console – The AWS Management Console provides a web interface for creating and managing APIs. After you complete the steps in Prerequisites), you can access the API Gateway console at https://console.aws.amazon.com/apigateway.

• AWS SDKs – If you're using a programming language that AWS provides an SDK for, you can use an SDK to access API Gateway. SDKs simplify authentication, integrate easily with your development environment, and provide access to API Gateway commands. For more information, see Tools for Amazon Web Services.

• API Gateway V1 and V2 APIs – If you're using a programming language that an SDK isn't available for, see the Amazon API Gateway Version 1 API Reference and Amazon API Gateway Version 2 API Reference.

• AWS Command Line Interface – For more information, see Getting Set Up with the AWS Command Line Interface in the AWS Command Line Interface User Guide.

• AWS Tools for Windows PowerShell – For more information, see Setting Up the AWS Tools for Windows PowerShell in the AWS Tools for Windows PowerShell User Guide.

Part of AWS serverless infrastructure

Together with AWS Lambda, API Gateway forms the app-facing part of the AWS serverless infrastructure.

For an app to call publicly available AWS services, you can use Lambda to interact with required services and expose Lambda functions through API methods in API Gateway. AWS Lambda runs your code on a highly available computing infrastructure. It performs the necessary execution and administration of computing resources. To enable serverless applications, API Gateway supports streamlined proxy integrations () with AWS Lambda and HTTP endpoints.

What is Amazon Cognito?

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. Your users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, Google or Apple. The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.

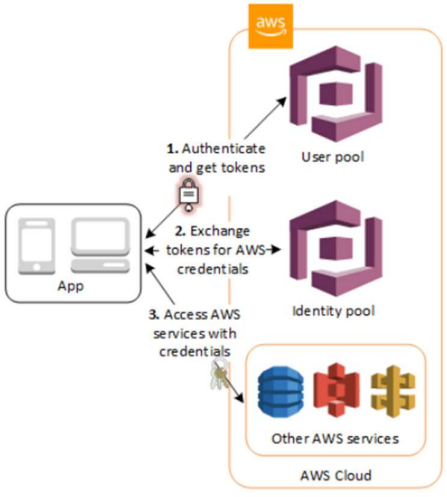
An Amazon Cognito user pool and identity pool used together

See the diagram for a common Amazon Cognito scenario. Here the goal is to authenticate your user, and then grant your user access to another AWS service.

1. In the first step your app user signs in through a user pool and receives user pool tokens after a successful authentication.

2. Next, your app exchanges the user pool tokens for AWS credentials through an identity pool.

3. Finally, your app user can then use those AWS credentials to access other AWS services such as Amazon S3 or DynamoDB.



Features of Amazon Cognito

User pools

A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign into your web or mobile app through Amazon Cognito or federate through a third-party identity provider (IdP). Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through an SDK.

User pools provide:

• Sign-up and sign-in services.

• A built-in, customizable web UI to sign in users.

• Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple, and through SAML and OIDC identity providers from your user pool.

• User directory management and user profiles.

• Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification.

• Customized workflows and user migration through AWS Lambda triggers.

Common Amazon Cognito scenarios

This topic describes six common scenarios for using Amazon Cognito.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your web and mobile app users. Identity pools provide AWS credentials to grant your users access to other AWS services.

A user pool is a user directory in Amazon Cognito. Your app users can sign in either directly through a user pool or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and

Apple, and from OpenID Connect (OIDC) and SAML IdPs. Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through an SDK.

With an identity pool, your users can obtain temporary AWS credentials to access AWS services, such as Amazon S3 and DynamoDB. Identity pools support anonymous guest users, as well as federation through third-party IdPs.

Integrating Amazon Cognito with web and mobile apps

When new users discover your app, or when existing users return to it, their first task is to sign up or sign in. When you integrate Amazon Cognito with your client code, you connect your app to AWS resources that aid authentication and authorization workflows. For example, your app uses the Amazon Cognito API to create new users in your user pool, retrieve user pool tokens, and obtain temporary credentials from your identity pool. To integrate Amazon Cognito with your web or mobile app, use AWS SDKs and libraries.

Although Amazon Cognito offers visual tools such as AWS Management Console integration and the hosted UI, AWS has designed the service to work with your app code. You can only configure certain components of Amazon Cognito with the API or the AWS Command Line Interface. For example, you can only register a user for time-based one-time password (TOTP) multi-factor authentication (MFA) with a process that starts with Associate Software Token. Before you use Amazon Cognito authentication and authorization, choose an app platform, and prepare your code to integrate with the service.

Conclusion

The Amazon Cognito service is useful when an app developer doesn't have the time or resources to invest in building a login page UI and maintain user credentials in a database. It also provides several features such as MFA (Multi-

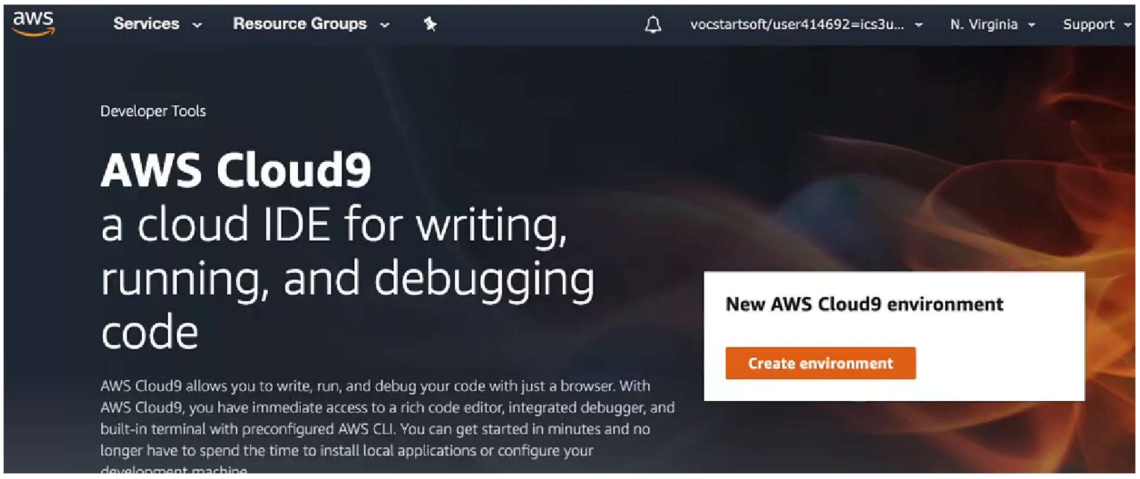
factor authentication), OTPs, prompts fingerprints or security questions. Phone numbers can be validated too.

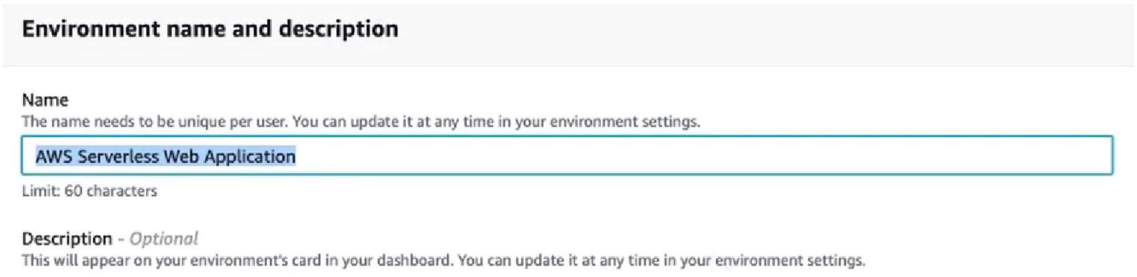
You are provided with an SDK – Amazon Cognito SDK where with only a few lines of code, you can set up a working user log-in page for your app. If your app already has a solution for user authentication, you can also migrate to Amazon Cognito with only a few steps. You can also prompt the users to sign up through Google, Facebook, Amazon, and other identity providers also.

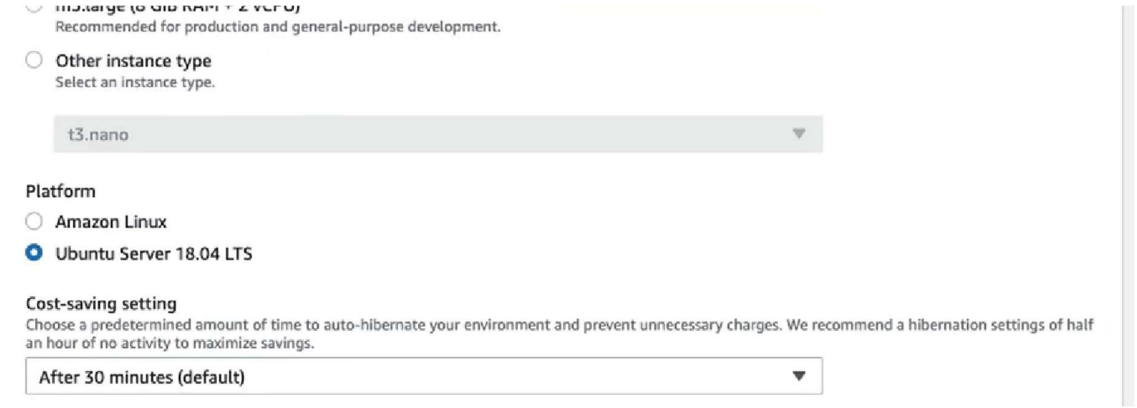
With all these benefits, the cost of the first 50000 MAF (Monthly active users is free). And it can scale itself as the user pool expands for a minimal amount per user after the free tier limit. So, this AWS service is very ideal for app developers with budget constraints and if they want their app to get going as soon as possible.

STEP 1

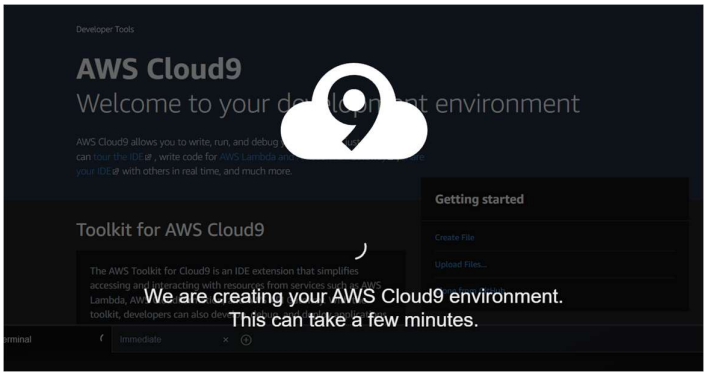
1. Create a Cloud9 instance



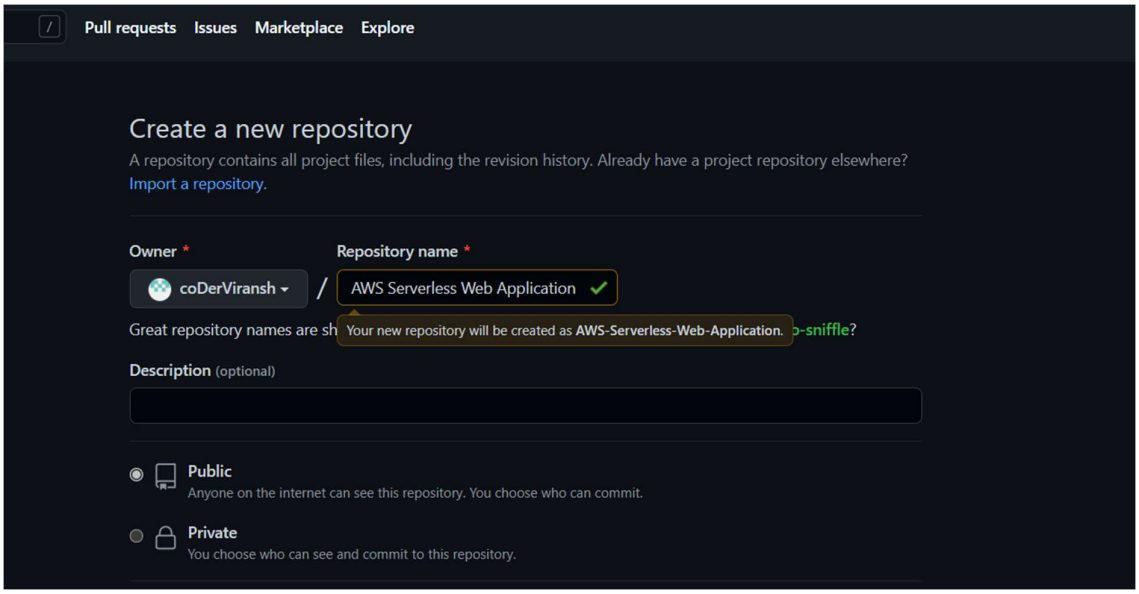
2. 

3. 

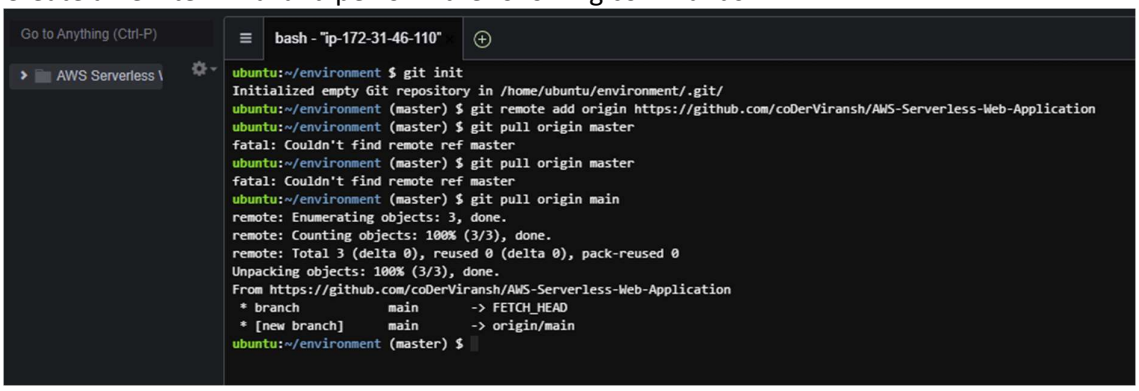
4. Create the environment and open github.



5. Create a repository in github and include a README File in the repository.



6. Click create the repository.
7. Copy the URL to the repository created to connect a cloud9 instance.
8. Open cloud9 environment and delete the README file.
9. Create a new terminal and perform the following commands.



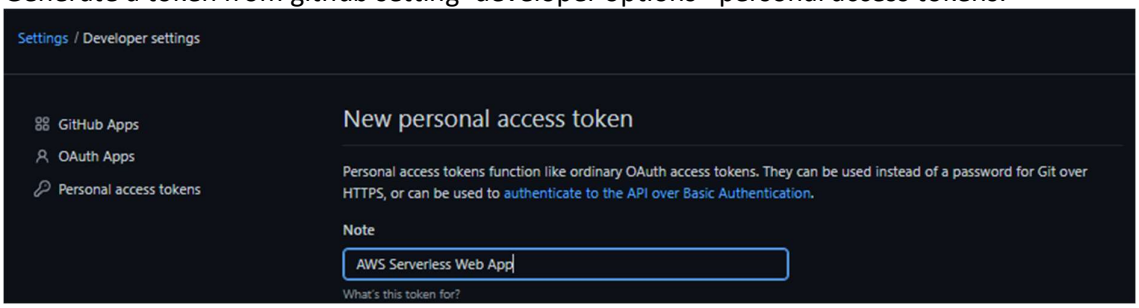
10. Create a new HTML file as follows and save the file as “index.html’.

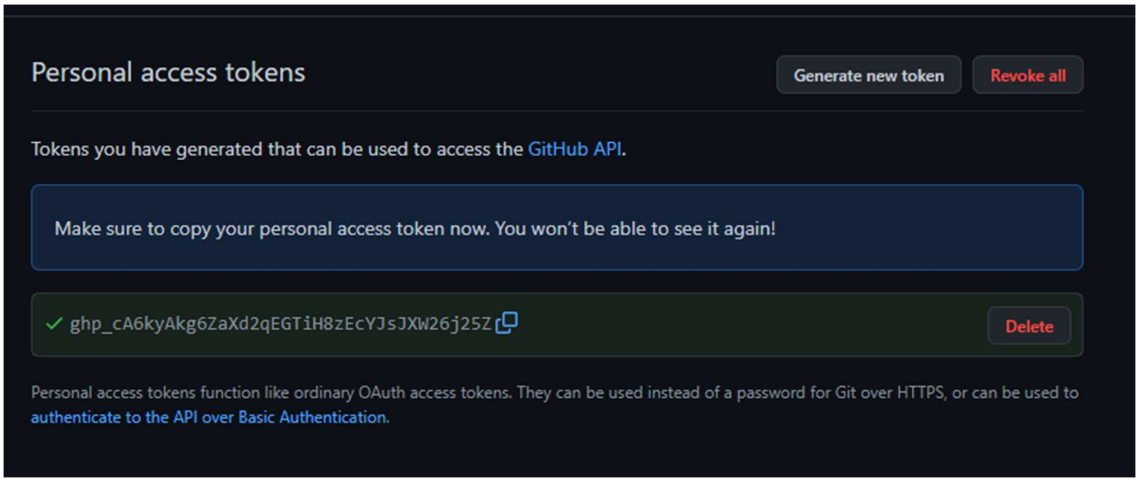
```html
<!DOCTYPE html>
<html>
  <head>
    <title> AWS Serverless Web App</title>
  </head>
  <body>
  Hello World!
  </body>
</html>
```

11. Perform the following commands in the ubuntu terminal in cloud9

```
ubuntu:~/environment (master) $ git add *
ubuntu:~/environment (master) $ git commit -m "added index.html"
[master ab10aa4] added index.html
 Committer: Ubuntu <ubuntu@ip-172-31-46-110.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 9 insertions(+)
 create mode 100644 index.html
ubuntu:~/environment (master) $ git config --global user.name "Group Project"
ubuntu:~/environment (master) $ git config --global user.mail viranshbhardwaj110203@gmail.com
```

12. Generate a token from github setting>developer options> personal access tokens.

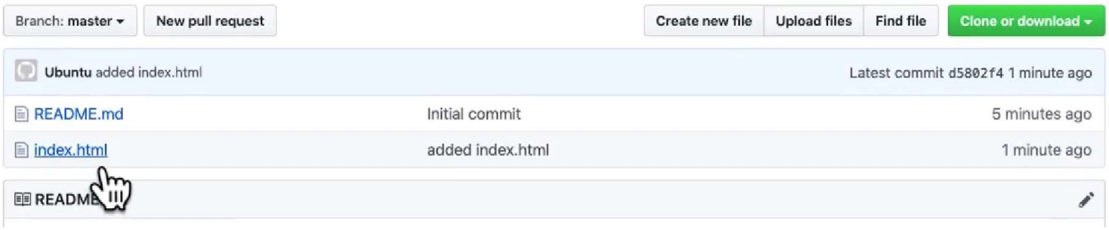


13. 

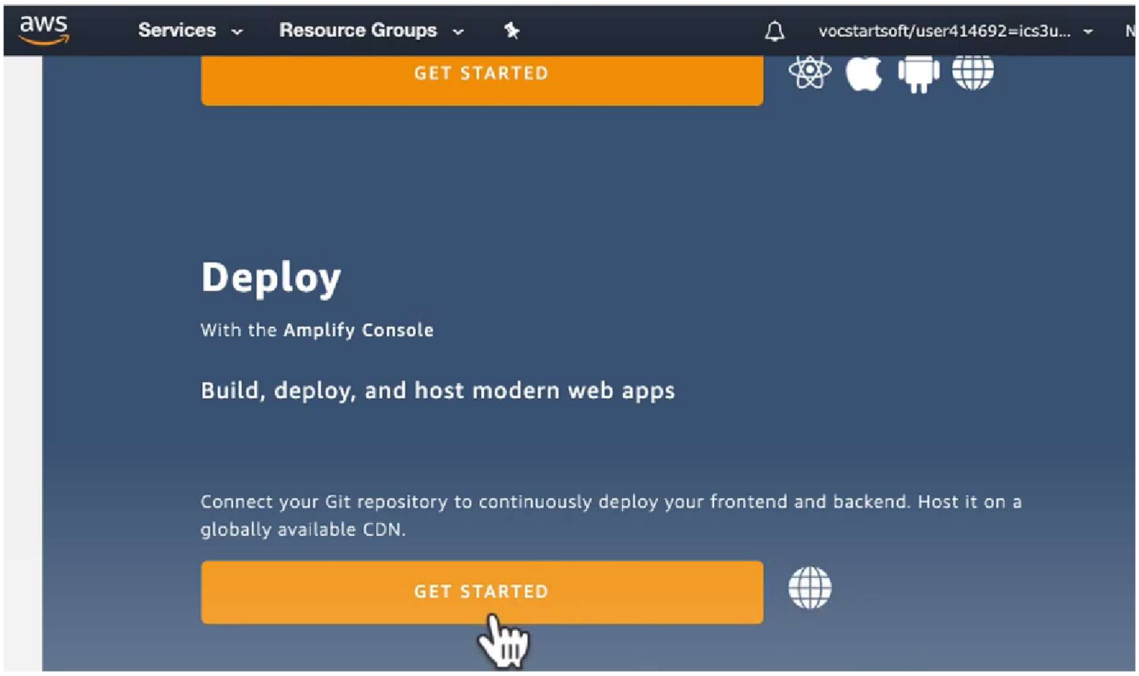
14. Furthermore, perform the following code.

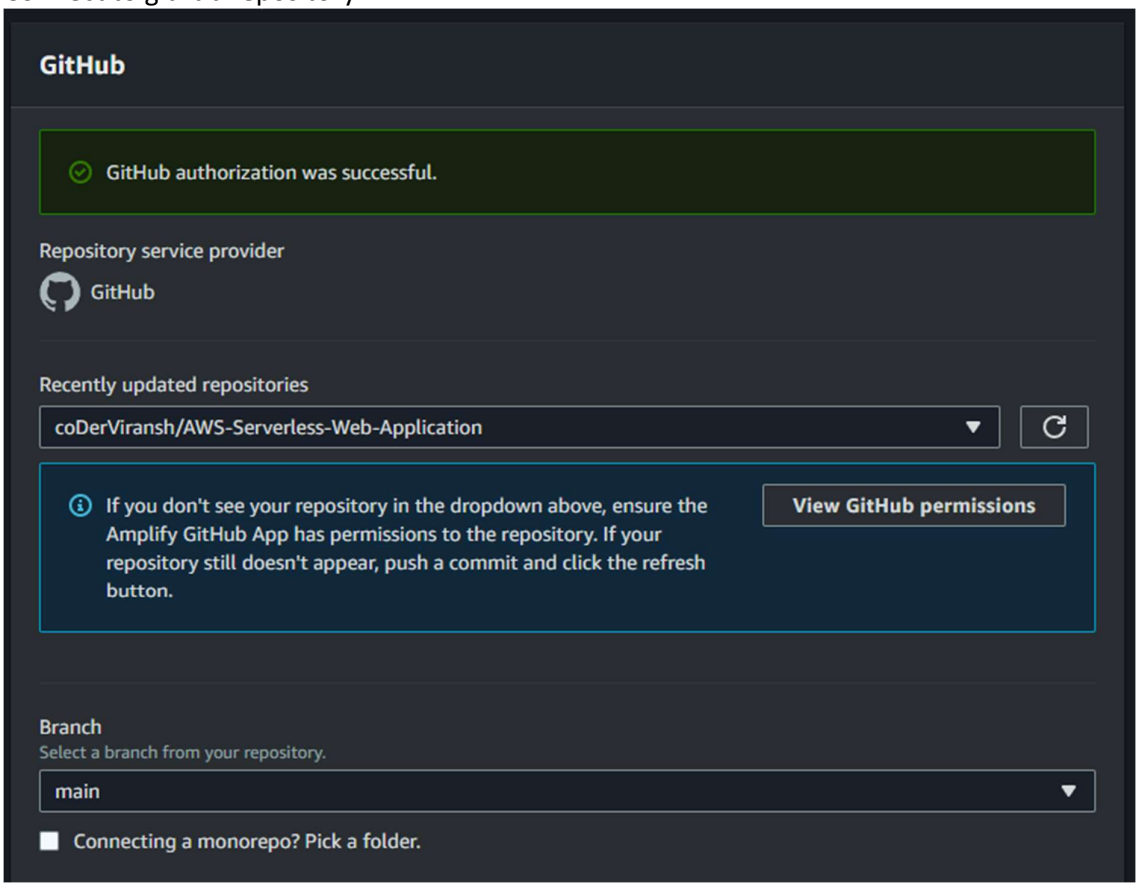
```
Username for 'https://github.com/coDerViransh/AWS-Serverless-Web-Application': GroupProject
Password for 'https://GroupProject@github.com/coDerViransh/AWS-Serverless-Web-Application':
```

15. Refresh your github repository to have the "index.html" file

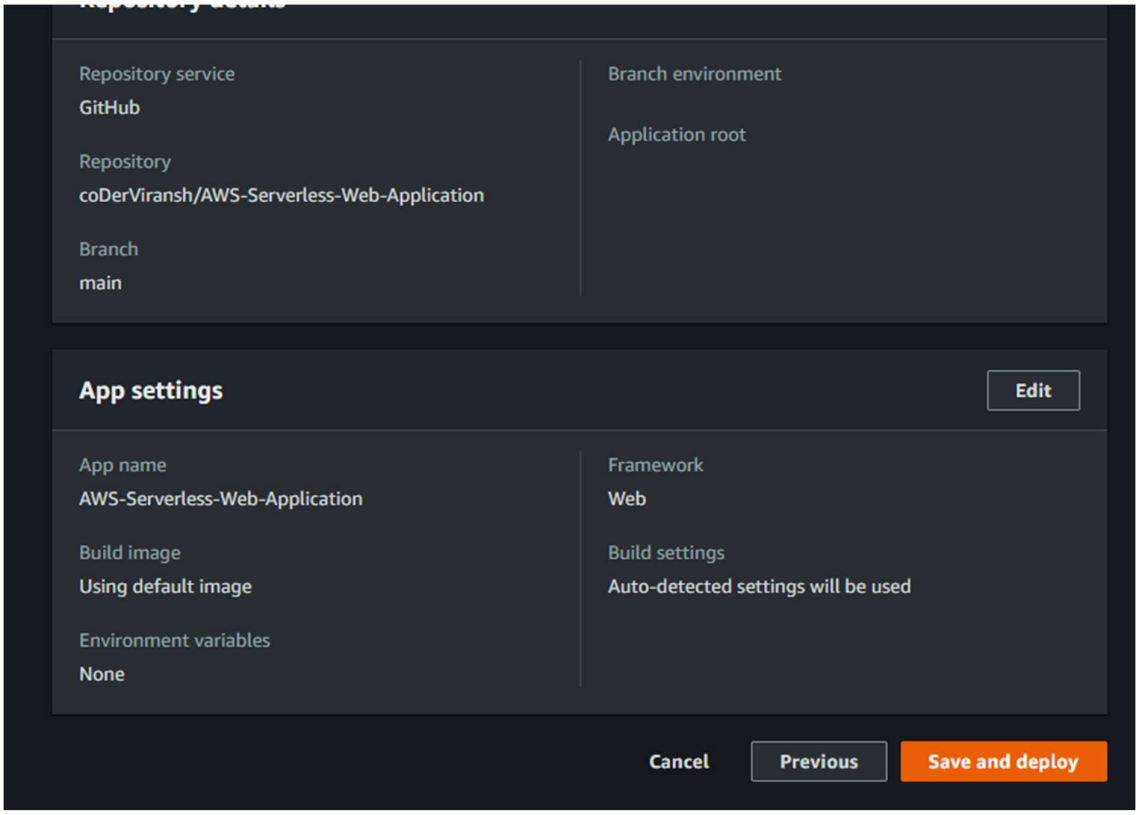


16. Open AWS Amplify console and chose the following option.

17. 

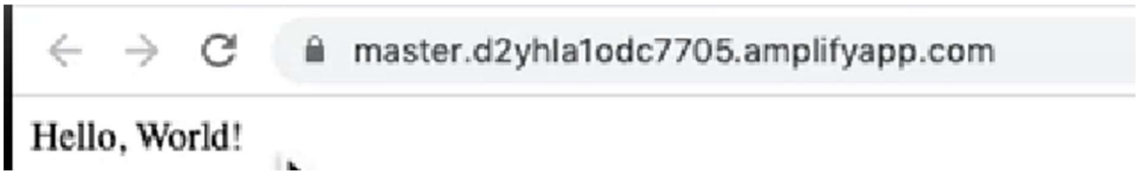
18. Connect to github repository.
19. 

20. Save and deploy without changing any other settings.

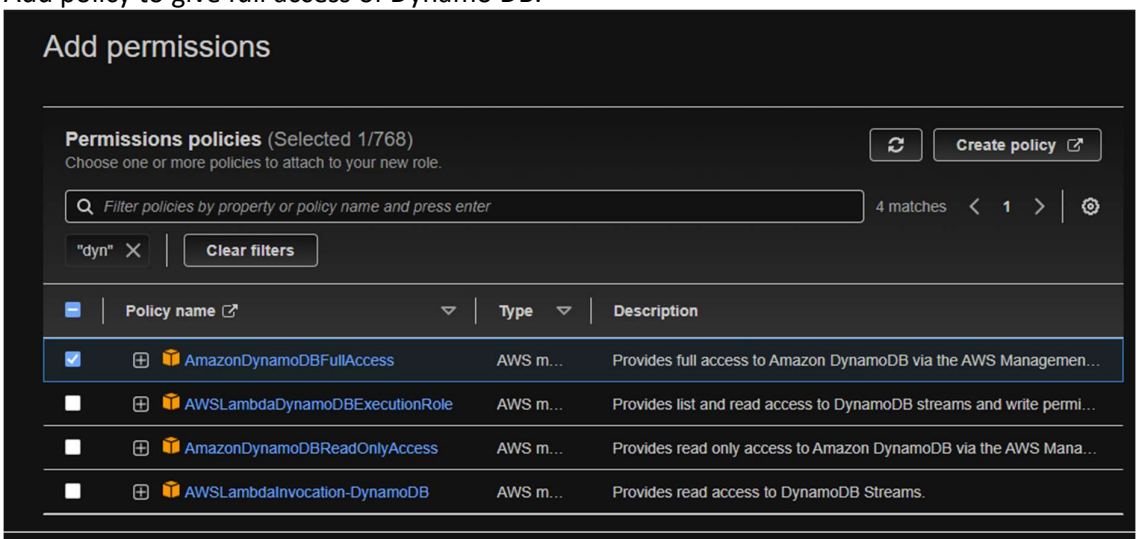
21. 

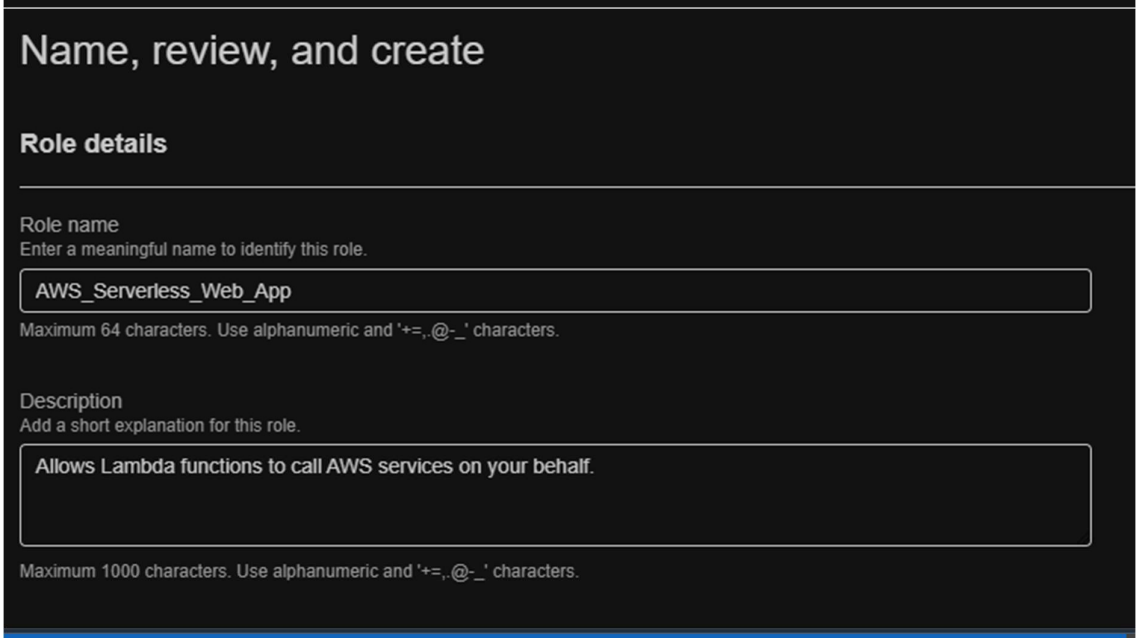
22. 

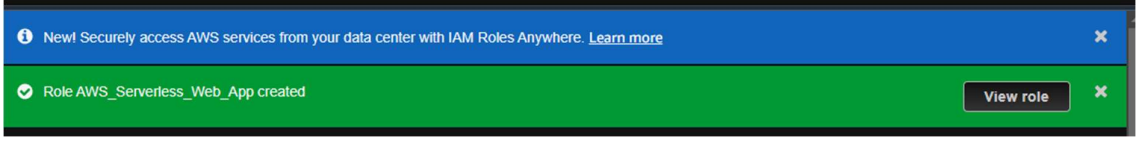
23. Click on the URL

24. 

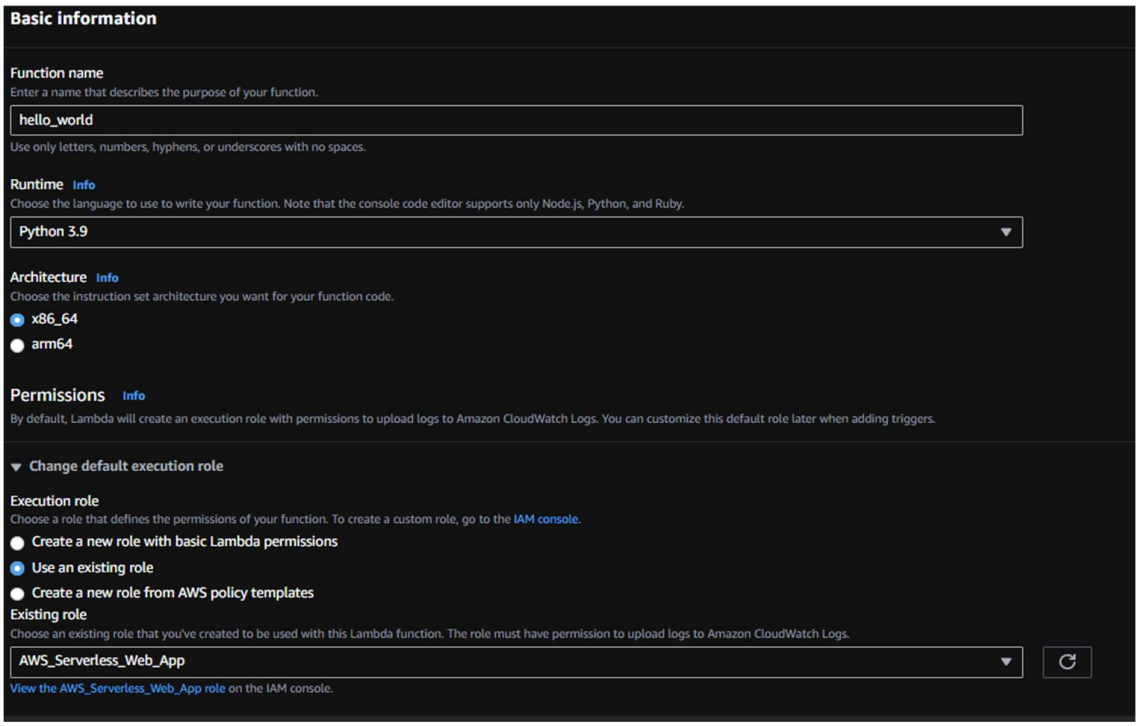
STEP 2

1. Open IAM
2. Create a new role.
3. Lambda function role
4. Add policy to give full access of Dynamo DB.
5. 

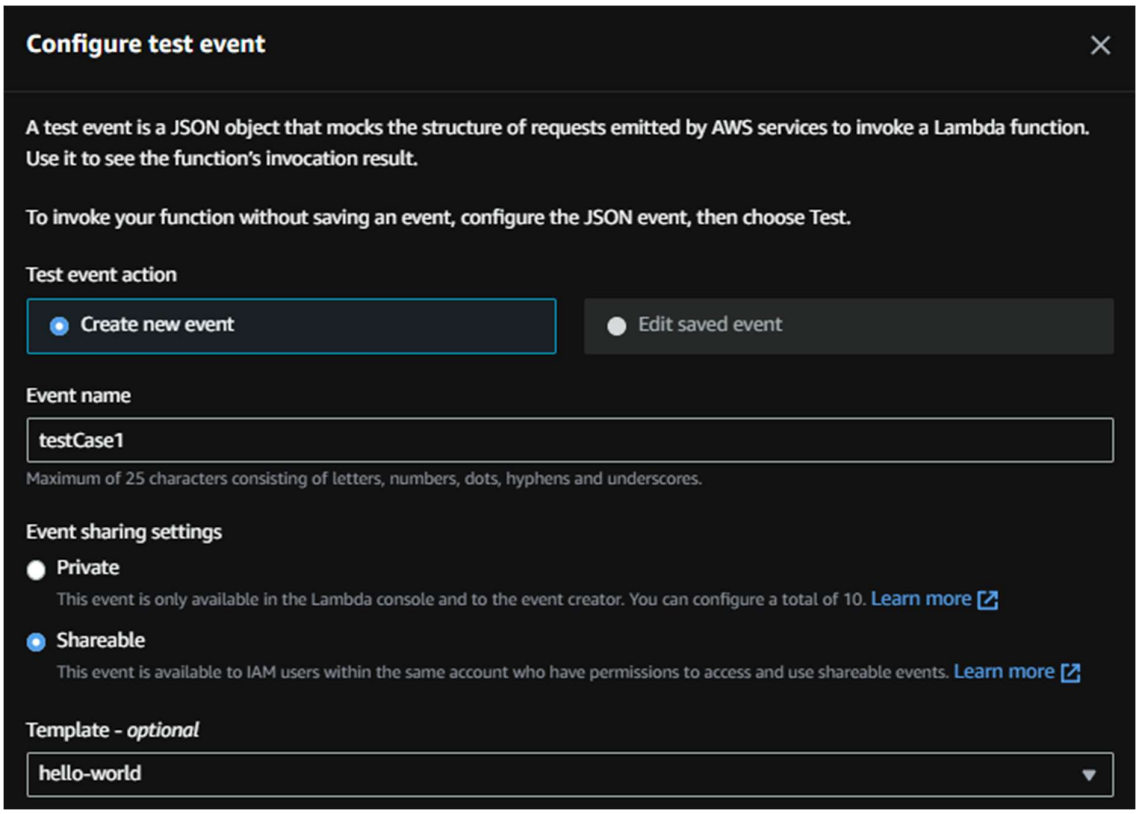
6. 

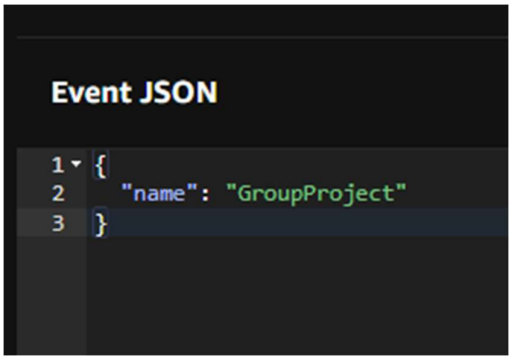
7. 

Step 3

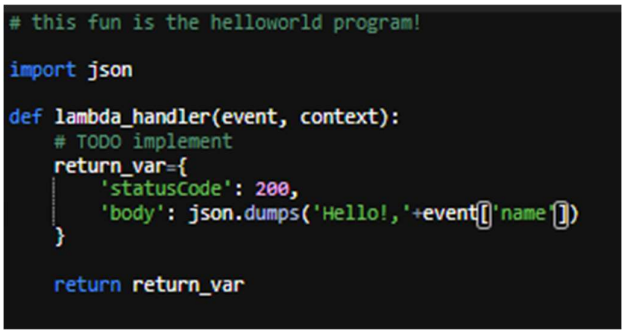
1. Open Lambda Service
2. Create a function from scratch.
3. 

4. In the lambda function type the following code.
5.
```python
# this fun is the helloworld program!

import json

def lambda_handler(event, context):
    # TODO implement
    return_var={
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }

    return return_var
```
6. Test the function.

7. 

8. 

Edit the test event.

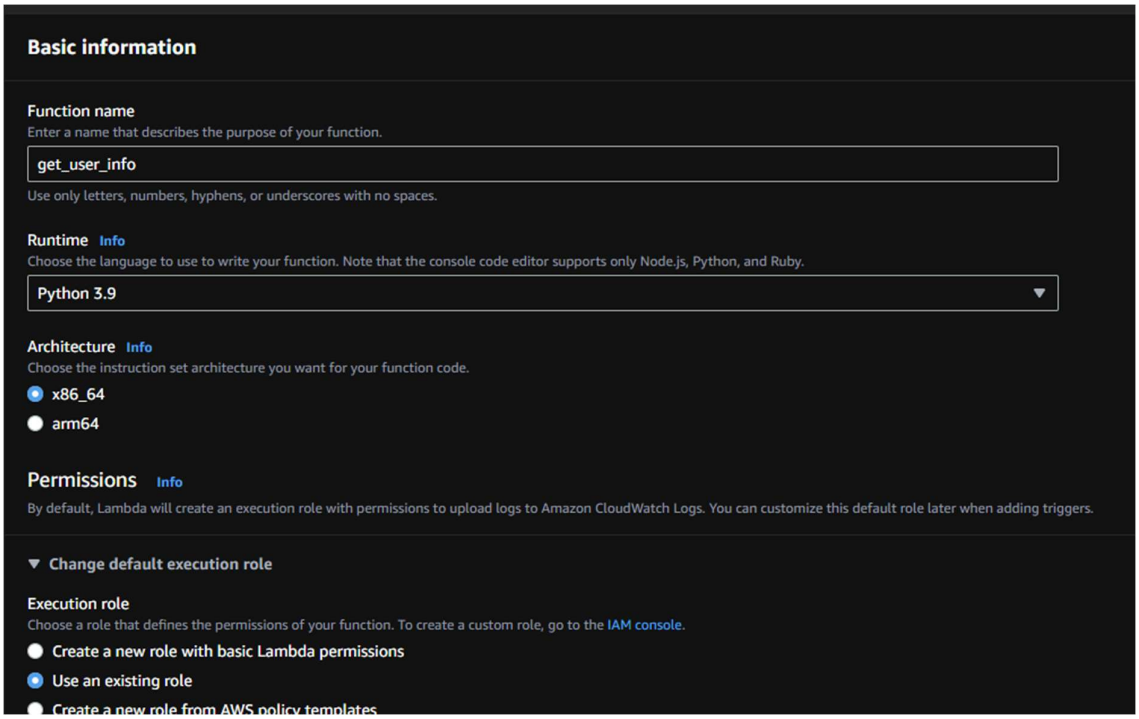
9. 

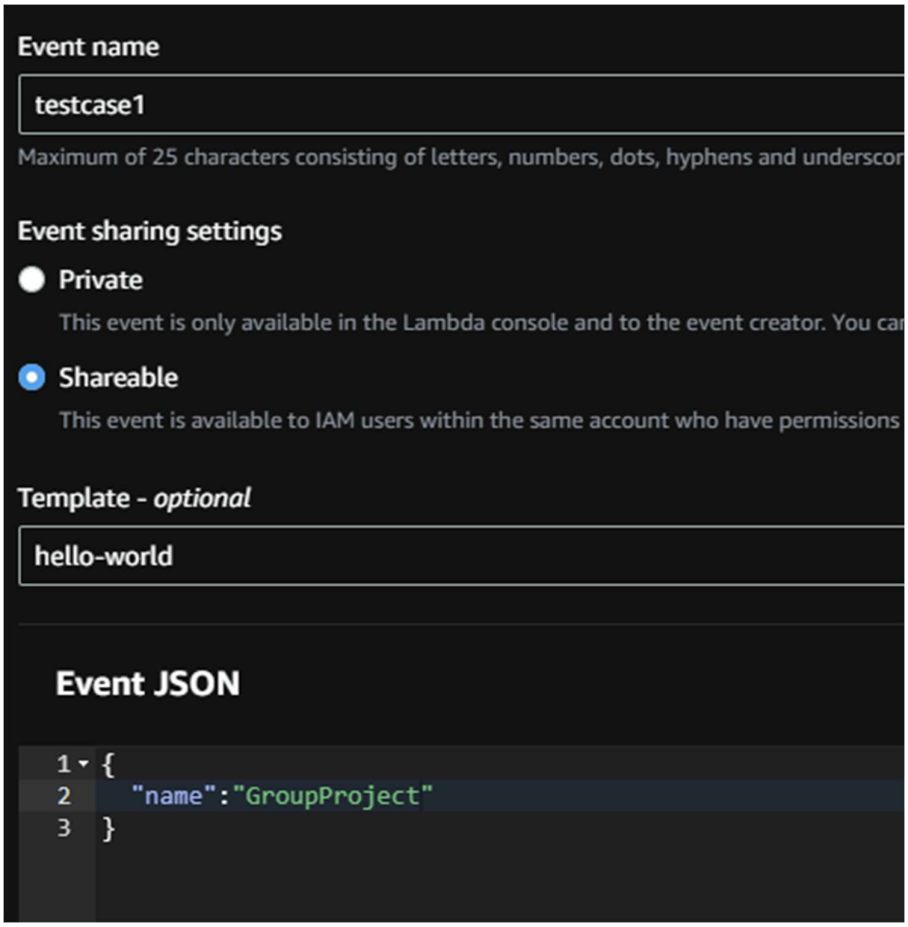
Edit the code.

10. Test Event

11. 

12. Similarly create another function “get_user_info” with the sanem code as before.

13. 

14. 

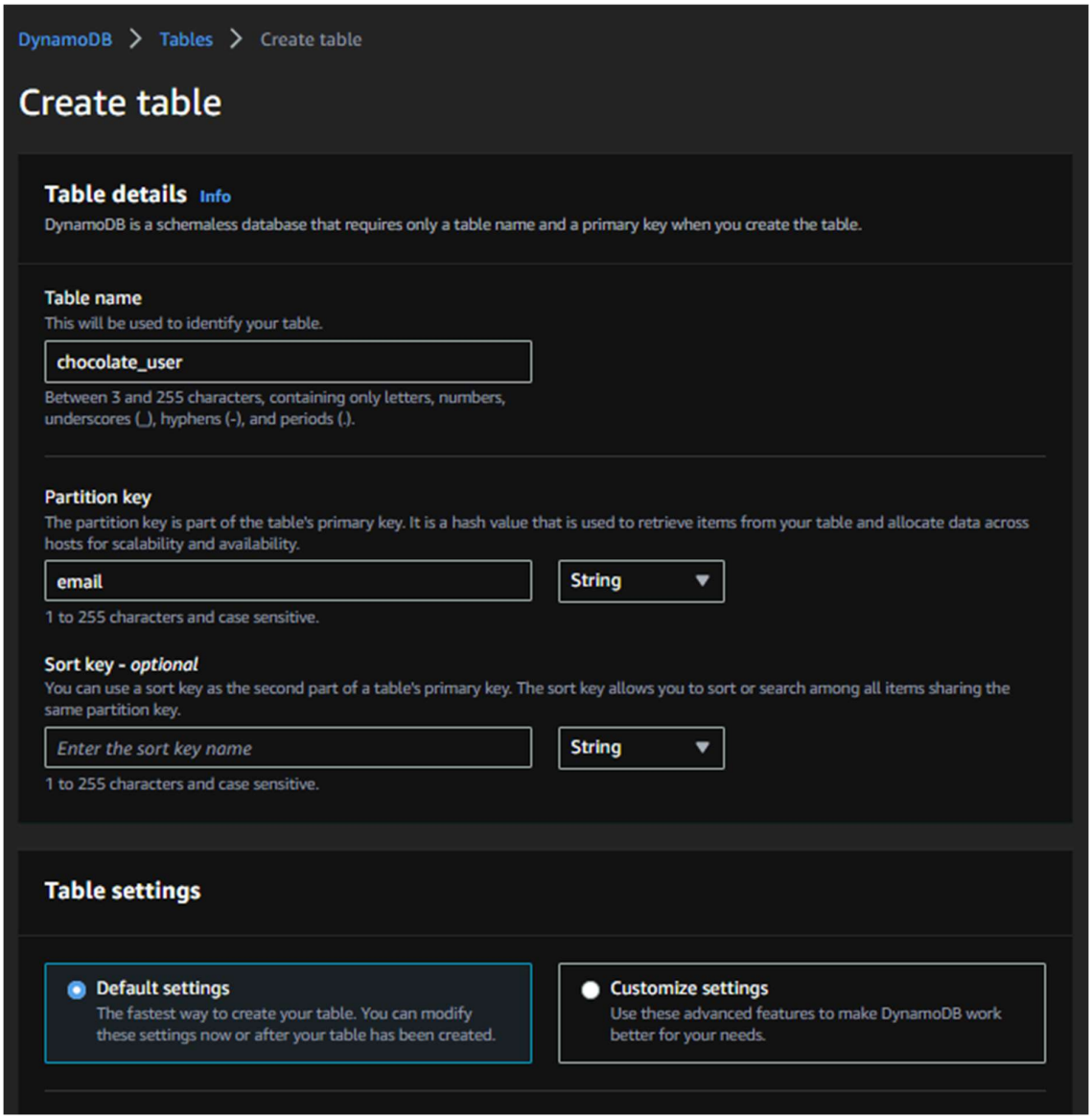
15. Type this code into the function.

```
# this function return a row from the dynamo DB table.


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)
```

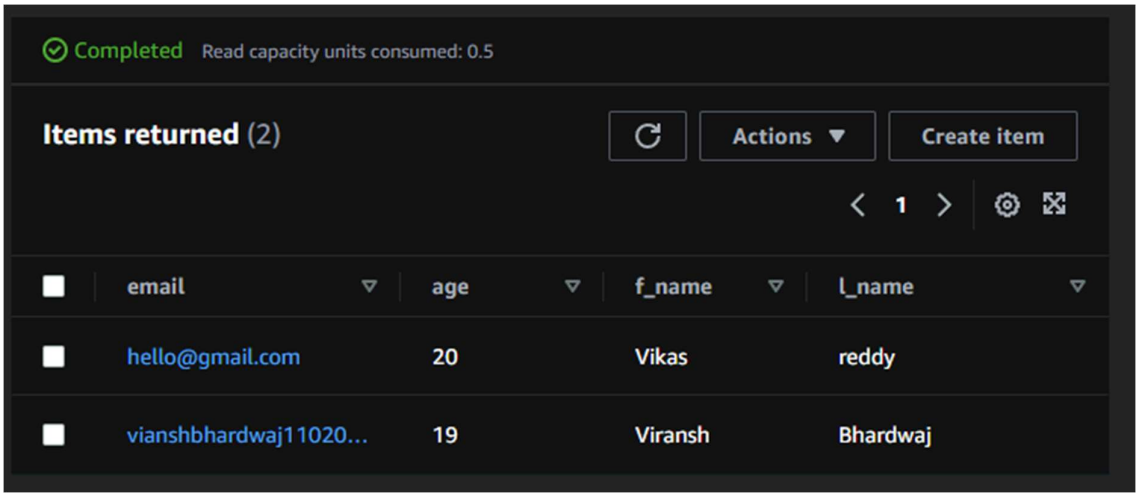
```
        return obj
    elif isinstance(obj, set):
        return set(replace_decimals(i) for i in obj)
    elif isinstance(obj, decimal.Decimal):
        if obj % 1 == 0:
            return int(obj)
        else:
            return float(obj)
    else:
        return obj

def lambda_handler(event, context):
    # TODO implement

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

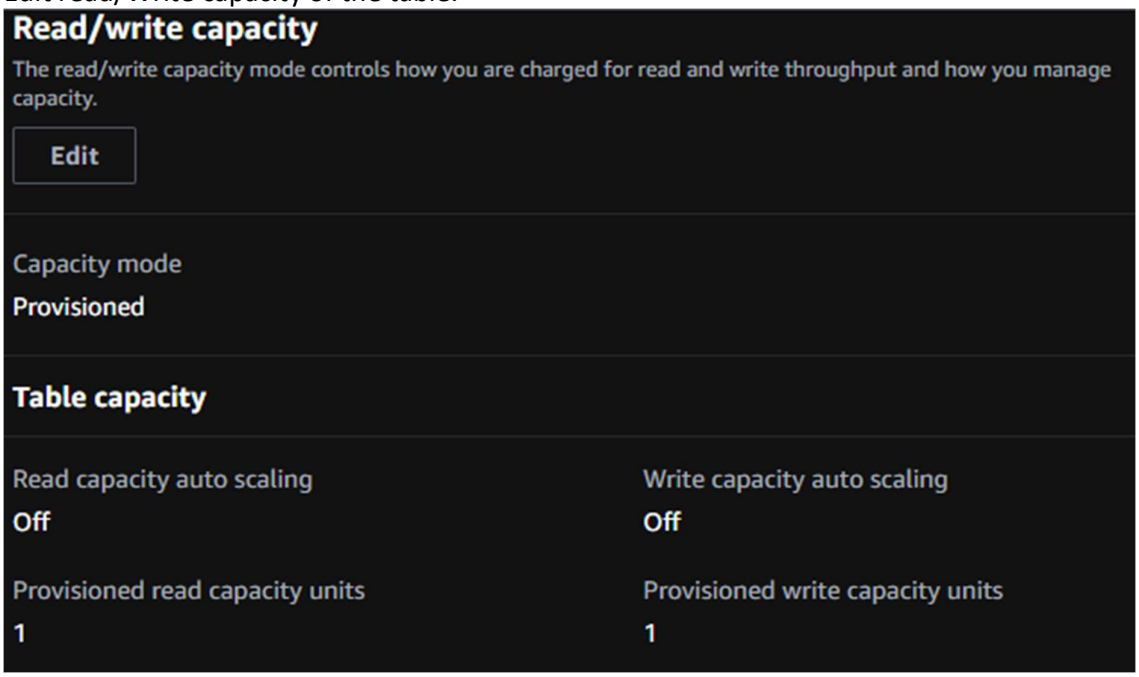
Step 4

1. Create table as follows.

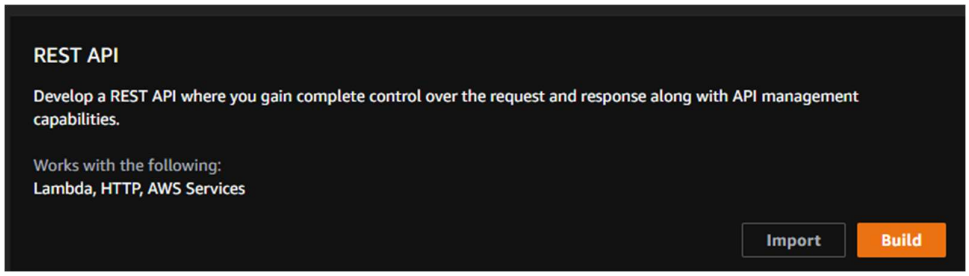


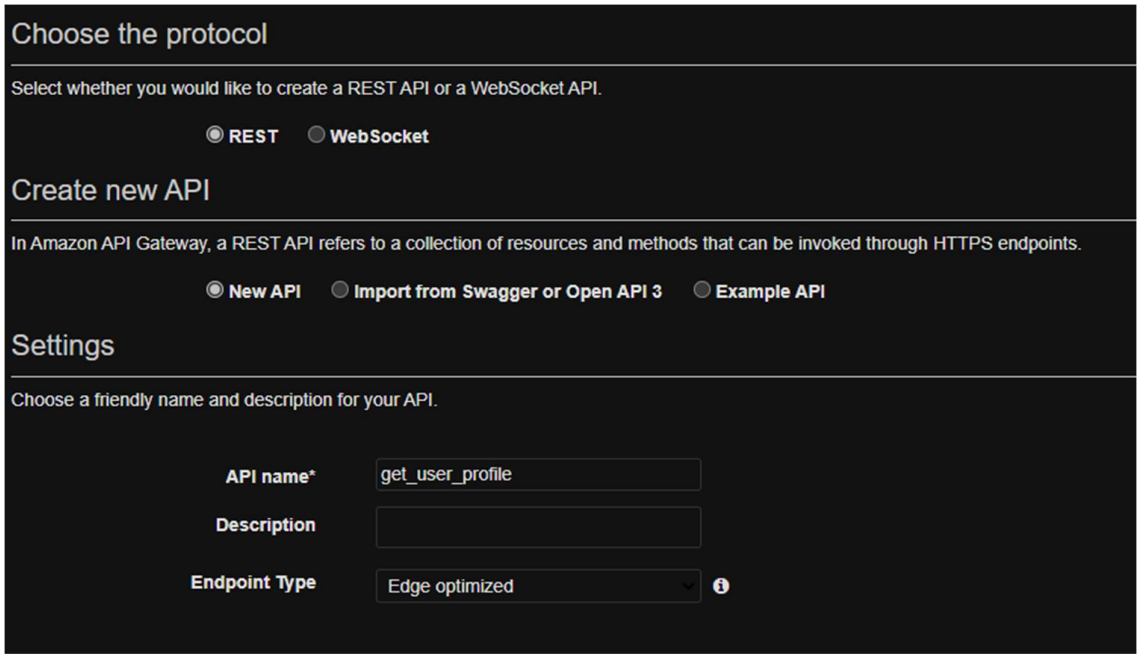
2. Create table items. As described.

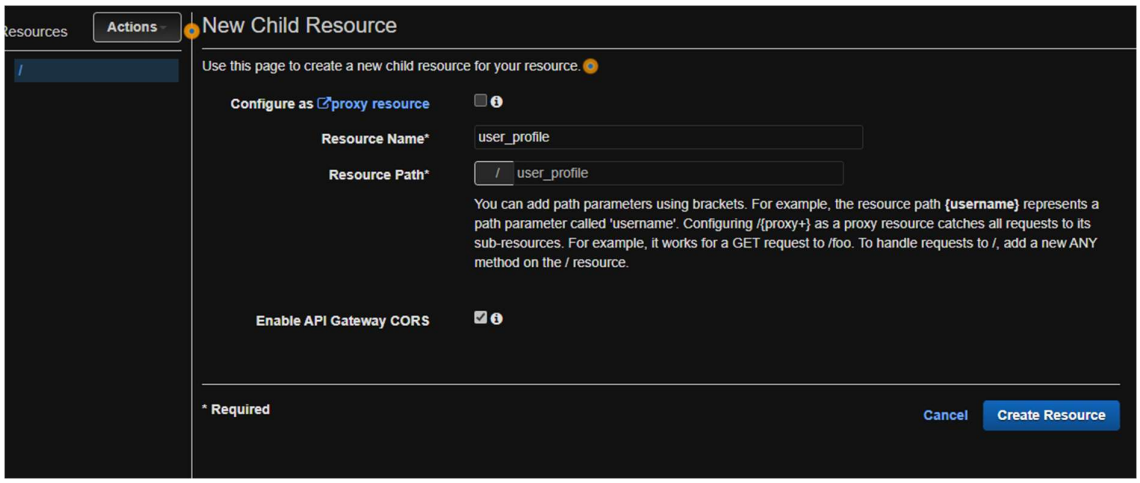
3. 

4. Edit read/Write capacity of the table.
5. 

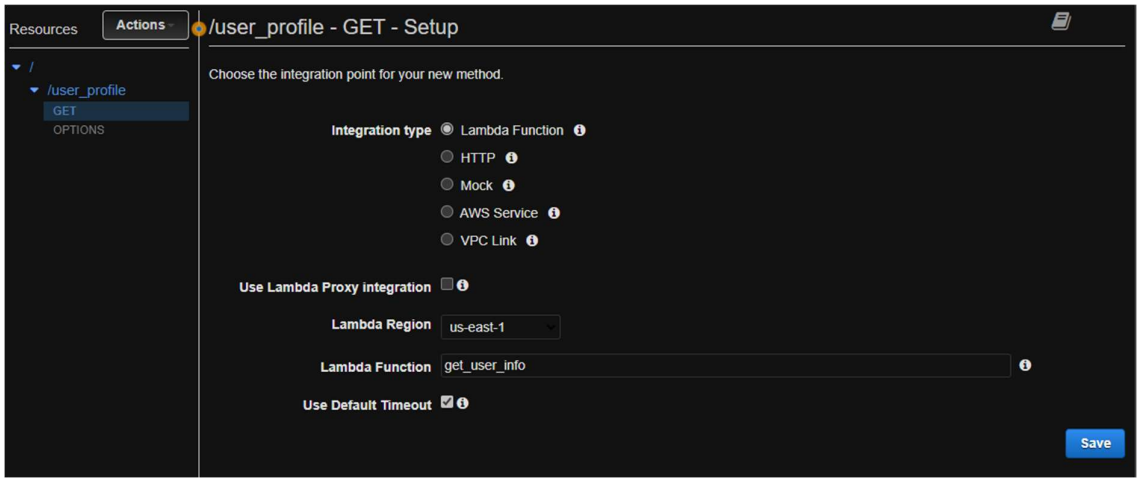
STEP 6

1. 
REST API
Develop a REST API where you gain complete control over the request and response along with API management capabilities.
Works with the following:
Lambda, HTTP, AWS Services
Import
Build

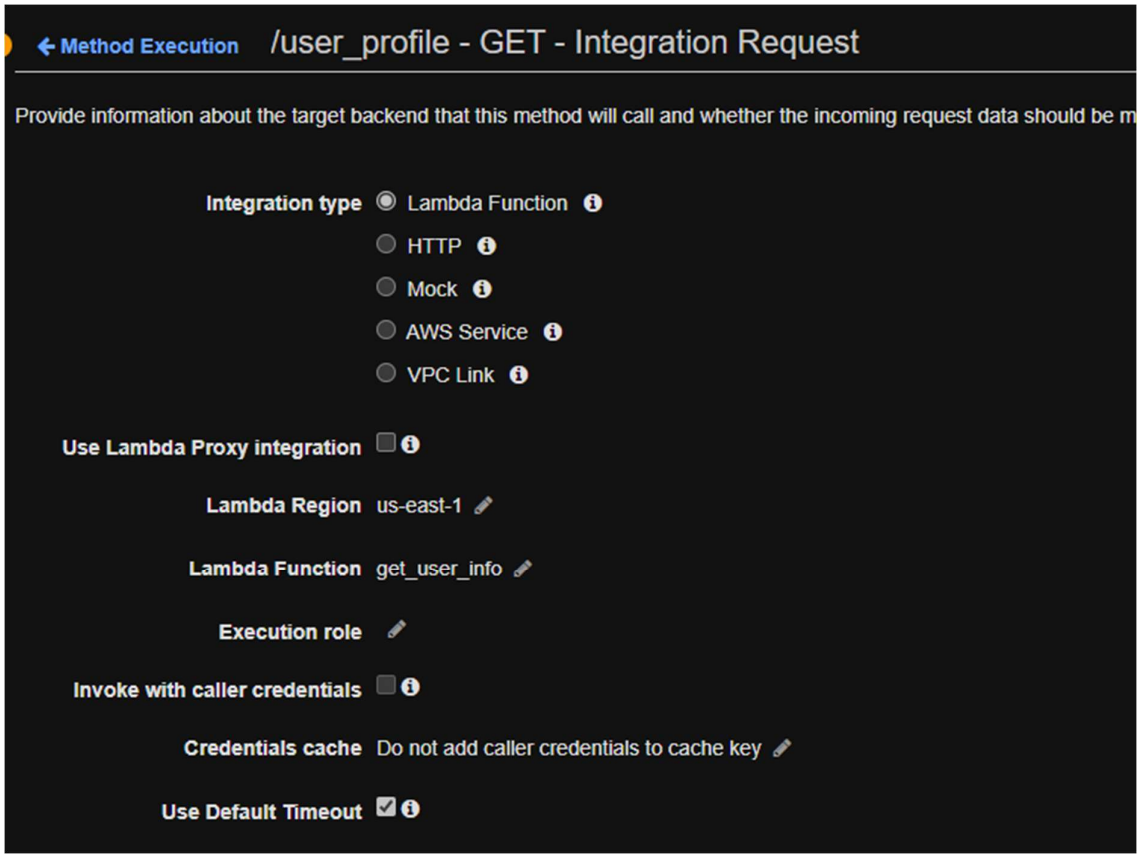
2. 
Choose the protocol
Select whether you would like to create a REST API or a WebSocket API.
REST
WebSocket
Create new API
In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.
New API
Import from Swagger or Open API 3
Example API
Settings
Choose a friendly name and description for your API.
API name*
get_user_profile
Description
Endpoint Type
Edge optimized

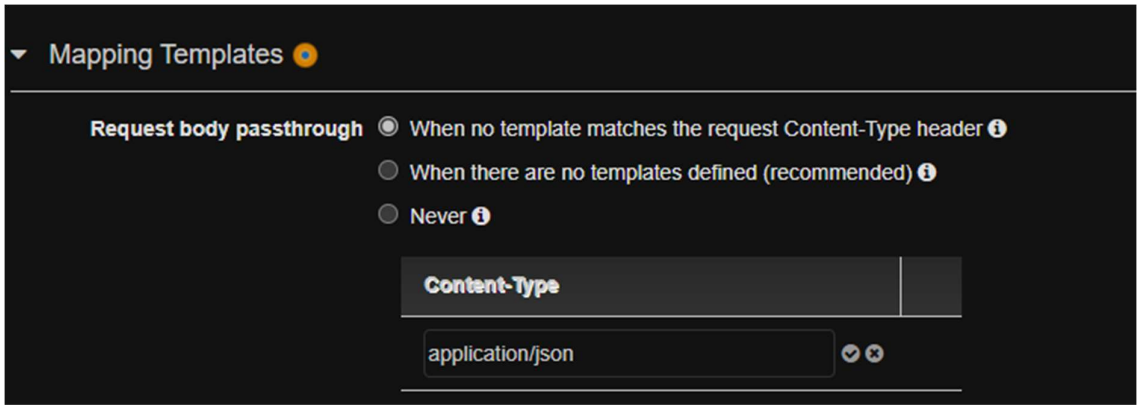
3. 
Resources
Actions
/
New Child Resource
Use this page to create a new child resource for your resource.
Configure as proxy resource
Resource Name*
user_profile
Resource Path*
/
user_profile
You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.
Enable API Gateway CORS
* Required
Cancel
Create Resource

4. Create a method GET and save.



5.

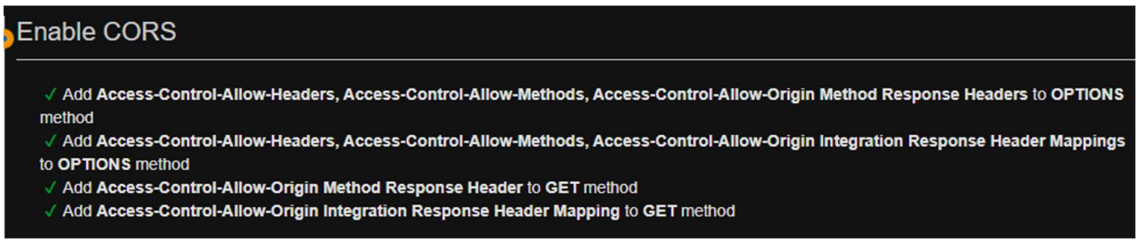


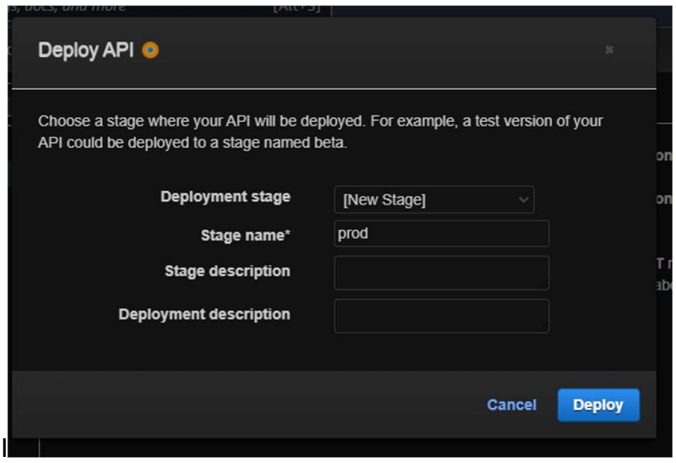
6. 

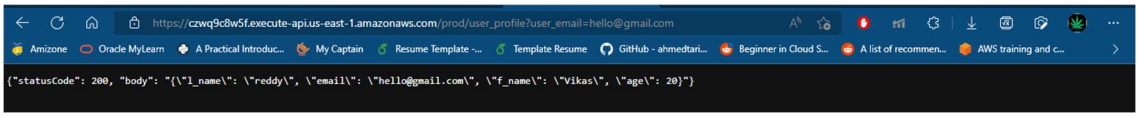
7. 

8. Test the method.

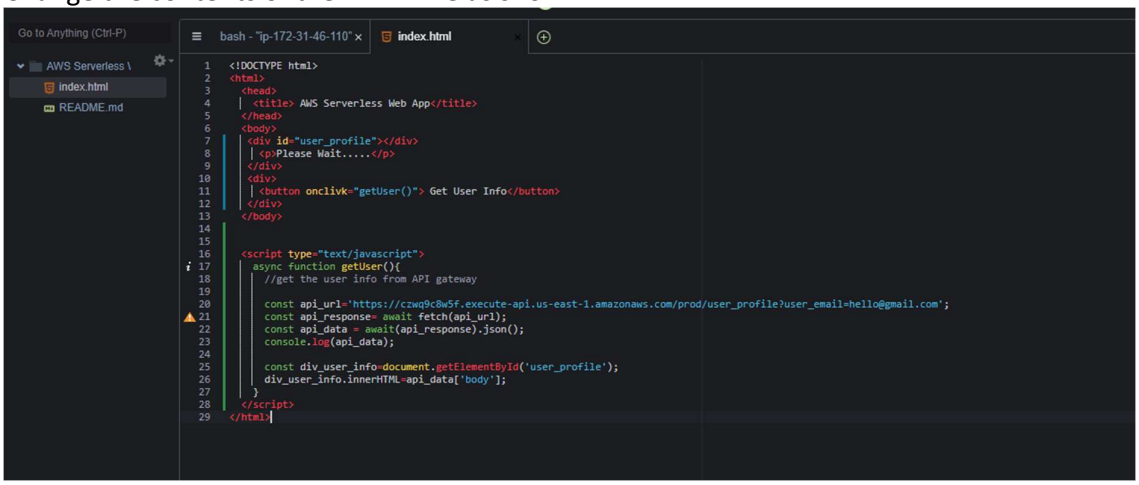
9. Enable CORS.

10. 

11. Deploy API 

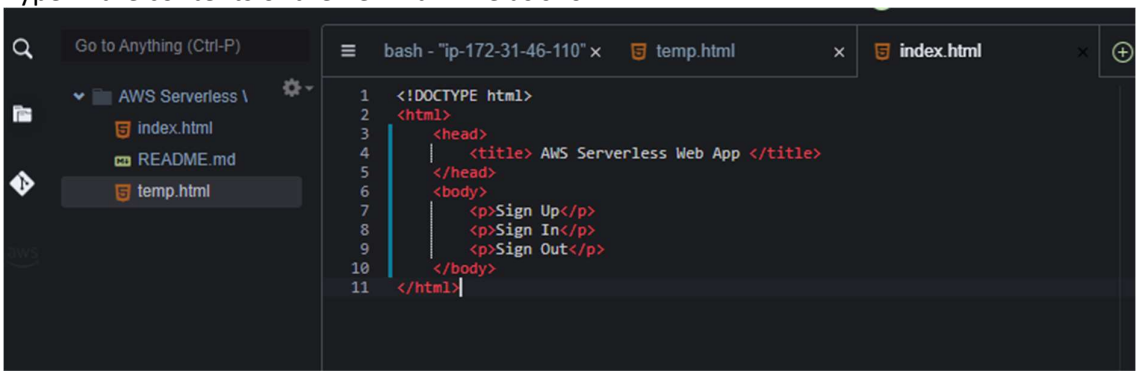
12. 

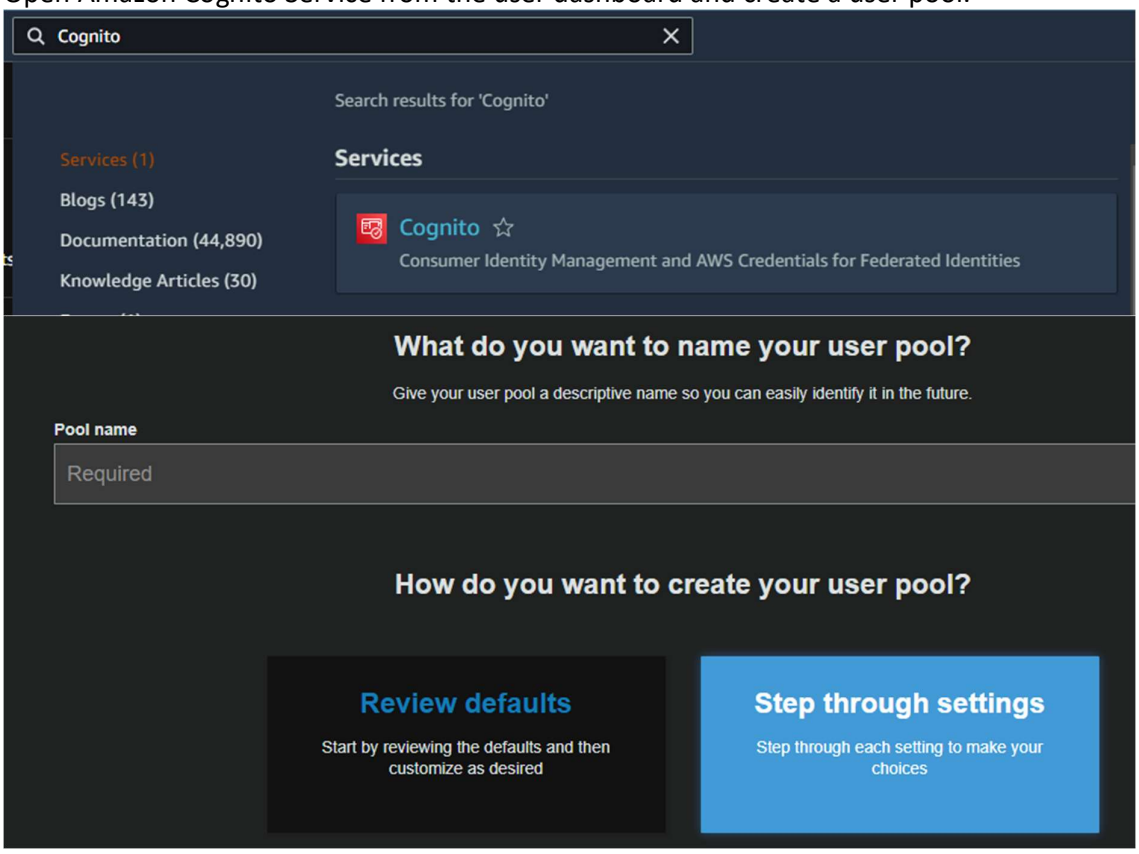
13. Change the contents of the HTML file as shown.



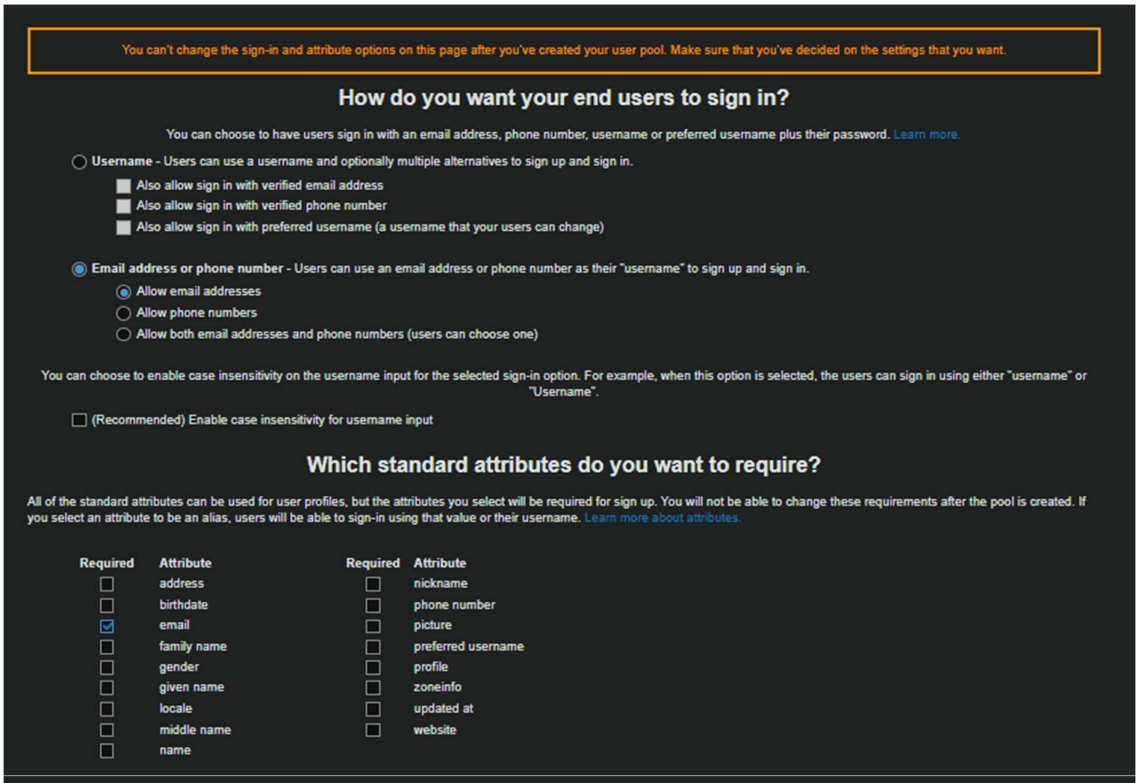
Step 7

1. Rename the html file as *temp.html* and create a new file as *index.html*
2. Type in the contents of the new html file as shown.

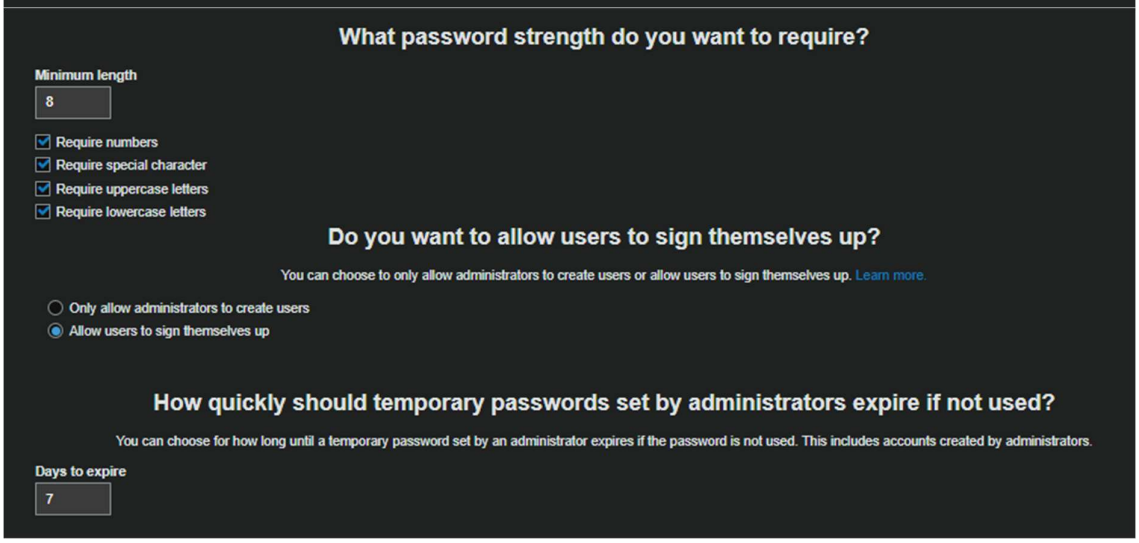


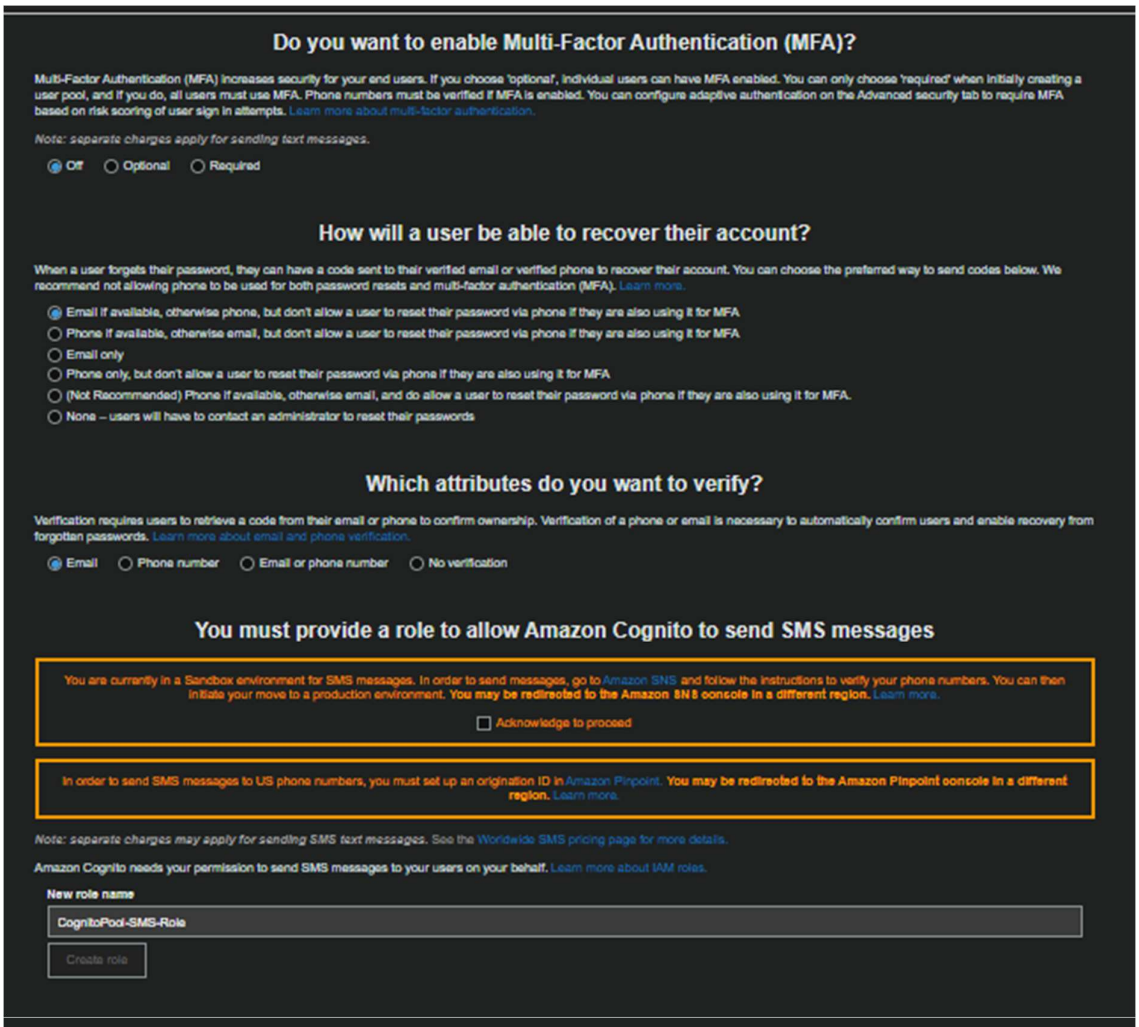
3. Open Amazon Cognito Service from the user dashboard and create a user pool.
4. 

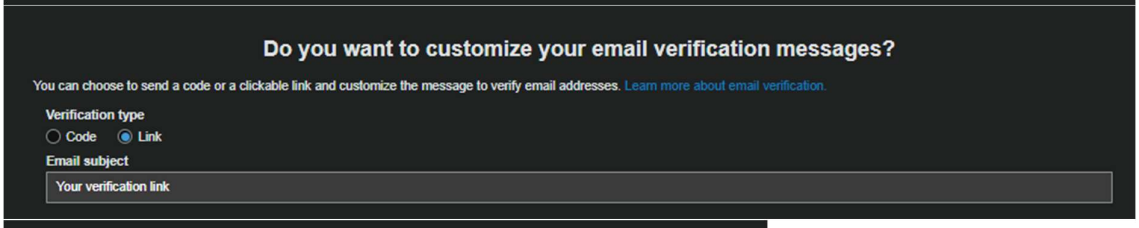
5.

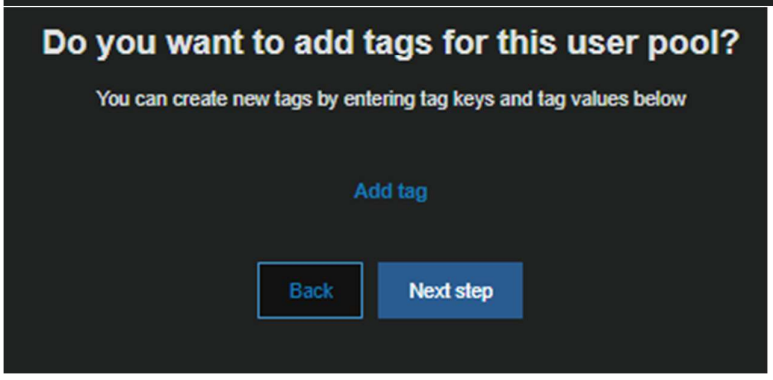

You can't change the sign-in and attribute options on this page after you've created your user pool. Make sure that you've decided on the settings that you want.
How do you want your end users to sign in?
You can choose to have users sign in with an email address, phone number, username or preferred username plus their password. Learn more.
Username - Users can use a username and optionally multiple alternatives to sign up and sign in.
Also allow sign in with verified email address
Also allow sign in with verified phone number
Also allow sign in with preferred username (a username that your users can change)
Email address or phone number - Users can use an email address or phone number as their "username" to sign up and sign in.
Allow email addresses
Allow phone numbers
Allow both email addresses and phone numbers (users can choose one)
You can choose to enable case insensitivity on the username input for the selected sign-in option. For example, when this option is selected, the users can sign in using either "username" or "Username".
(Recommended) Enable case insensitivity for username input
Which standard attributes do you want to require?
All of the standard attributes can be used for user profiles, but the attributes you select will be required for sign up. You will not be able to change these requirements after the pool is created. If you select an attribute to be an alias, users will be able to sign-in using that value or their username. Learn more about attributes.
Required
Attribute
address
birthdate
email
family name
gender
given name
locale
middle name
name
Required
Attribute
nickname
phone number
picture
preferred username
profile
zoneinfo
updated at
website

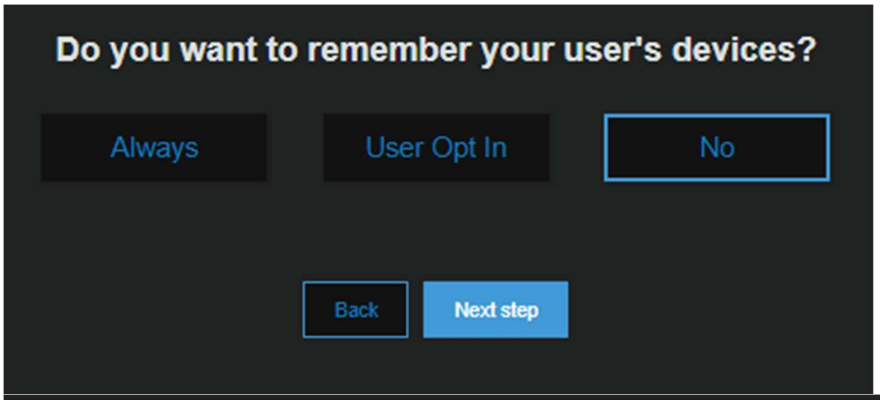
6.

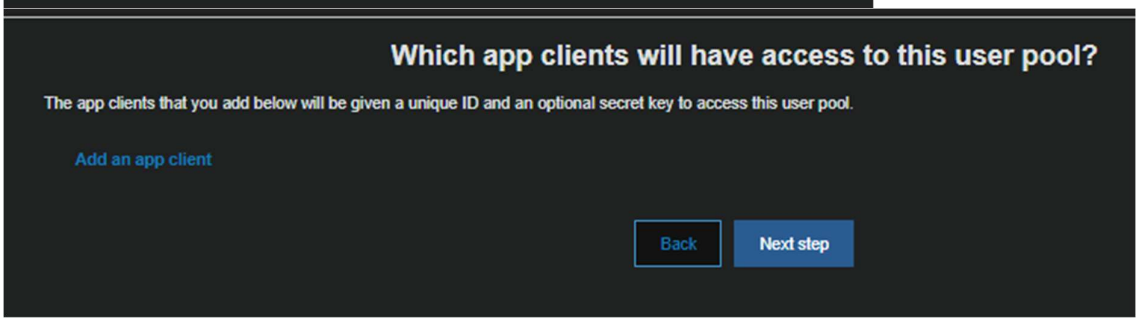

What password strength do you want to require?
Minimum length
8
Require numbers
Require special character
Require uppercase letters
Require lowercase letters
Do you want to allow users to sign themselves up?
You can choose to only allow administrators to create users or allow users to sign themselves up. Learn more.
Only allow administrators to create users
Allow users to sign themselves up
How quickly should temporary passwords set by administrators expire if not used?
You can choose for how long until a temporary password set by an administrator expires if the password is not used. This includes accounts created by administrators.
Days to expire
7

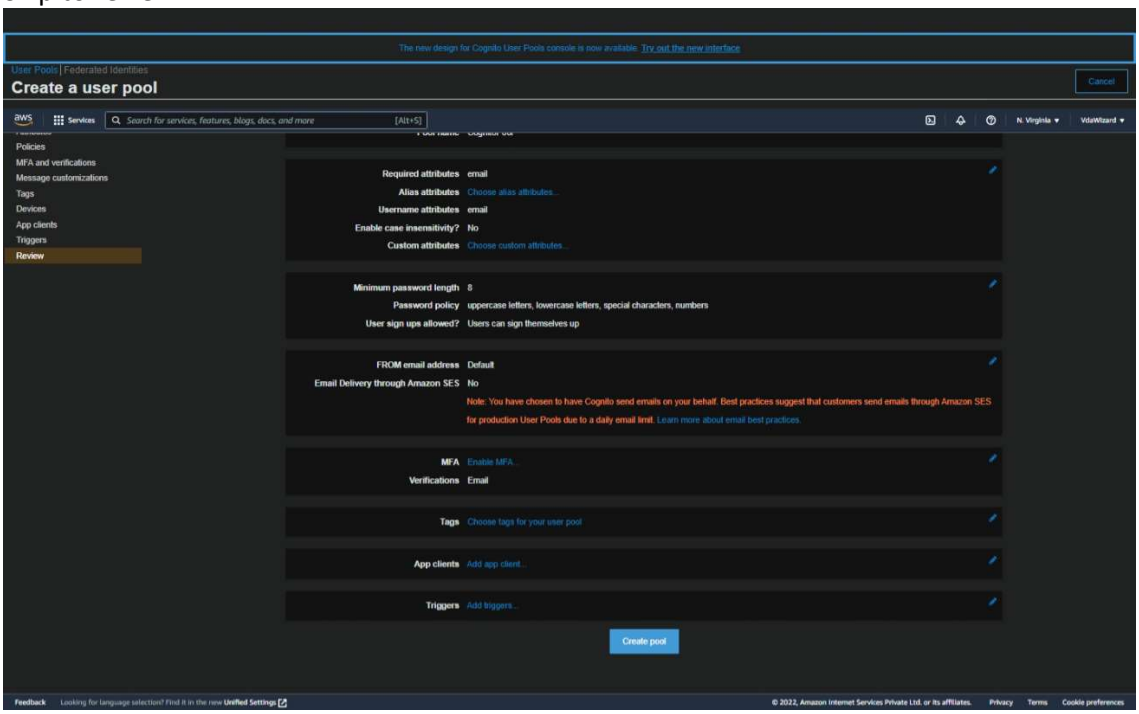
7. 
Do you want to enable Multi-Factor Authentication (MFA)?
Multi-Factor Authentication (MFA) increases security for your end users. If you choose 'optional', individual users can have MFA enabled. You can only choose 'required' when initially creating a user pool, and if you do, all users must use MFA. Phone numbers must be verified if MFA is enabled. You can configure adaptive authentication on the Advanced security tab to require MFA based on risk scoring of user sign in attempts. Learn more about multi-factor authentication.
Note: separate charges apply for sending text messages.
Off
Optional
Required
How will a user be able to recover their account?
When a user forgets their password, they can have a code sent to their verified email or verified phone to recover their account. You can choose the preferred way to send codes below. We recommend not allowing phone to be used for both password resets and multi-factor authentication (MFA). Learn more.
Email if available, otherwise phone, but don't allow a user to reset their password via phone if they are also using it for MFA
Phone if available, otherwise email, but don't allow a user to reset their password via phone if they are also using it for MFA
Email only
Phone only, but don't allow a user to reset their password via phone if they are also using it for MFA
(Not Recommended) Phone if available, otherwise email, and do allow a user to reset their password via phone if they are also using it for MFA.
None – users will have to contact an administrator to reset their passwords
Which attributes do you want to verify?
Verification requires users to retrieve a code from their email or phone to confirm ownership. Verification of a phone or email is necessary to automatically confirm users and enable recovery from forgotten passwords. Learn more about email and phone verification.
Email
Phone number
Email or phone number
No verification
You must provide a role to allow Amazon Cognito to send SMS messages
You are currently in a Sandbox environment for SMS messages. In order to send messages, go to Amazon SNS and follow the instructions to verify your phone numbers. You can then initiate your move to a production environment. You may be redirected to the Amazon SNS console in a different region. Learn more.
Acknowledge to proceed
In order to send SMS messages to US phone numbers, you must set up an origination ID in Amazon Pinpoint. You may be redirected to the Amazon Pinpoint console in a different region. Learn more.
Note: separate charges may apply for sending SMS text messages. See the Worldwide SMS pricing page for more details.
Amazon Cognito needs your permission to send SMS messages to your users on your behalf. Learn more about IAM roles.
New role name
CognitoPool-SMS-Role
Create role

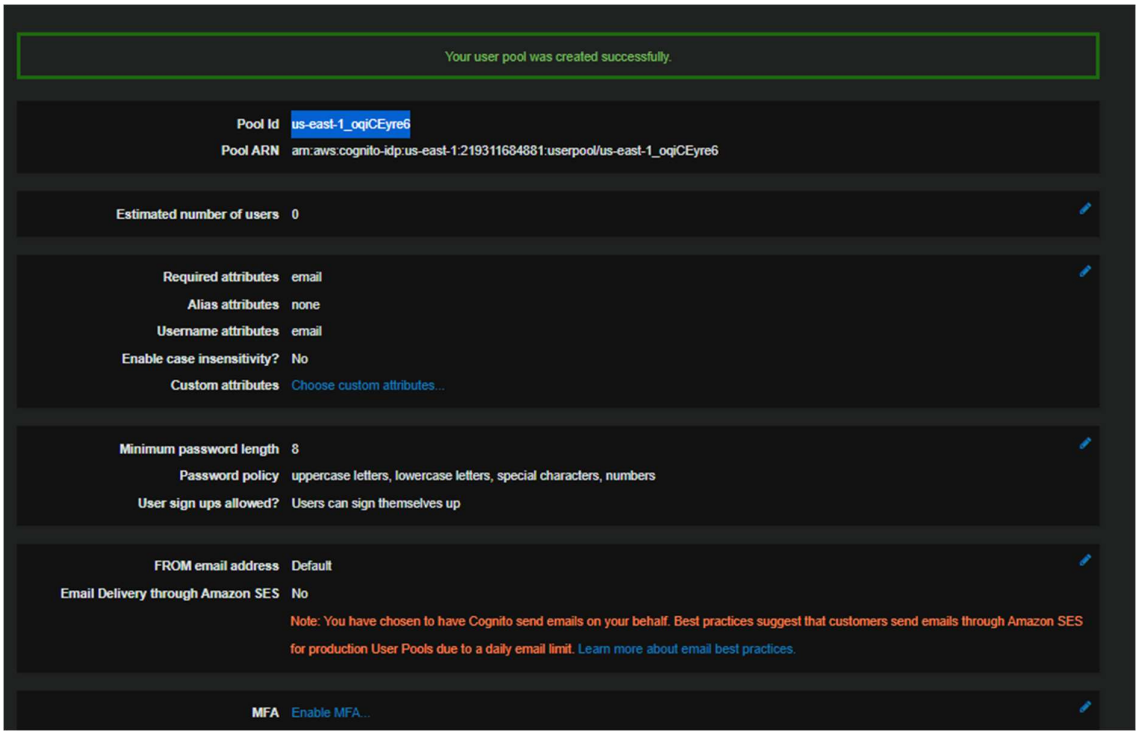
8. 
Do you want to customize your email verification messages?
You can choose to send a code or a clickable link and customize the message to verify email addresses. Learn more about email verification.
Verification type
Code
Link
Email subject
Your verification link

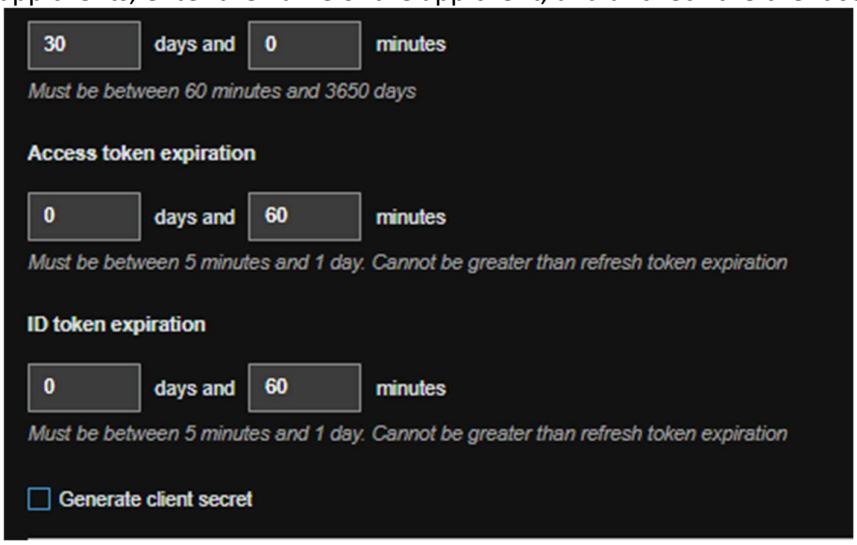
9. 
Do you want to add tags for this user pool?
You can create new tags by entering tag keys and tag values below
Add tag
Back
Next step

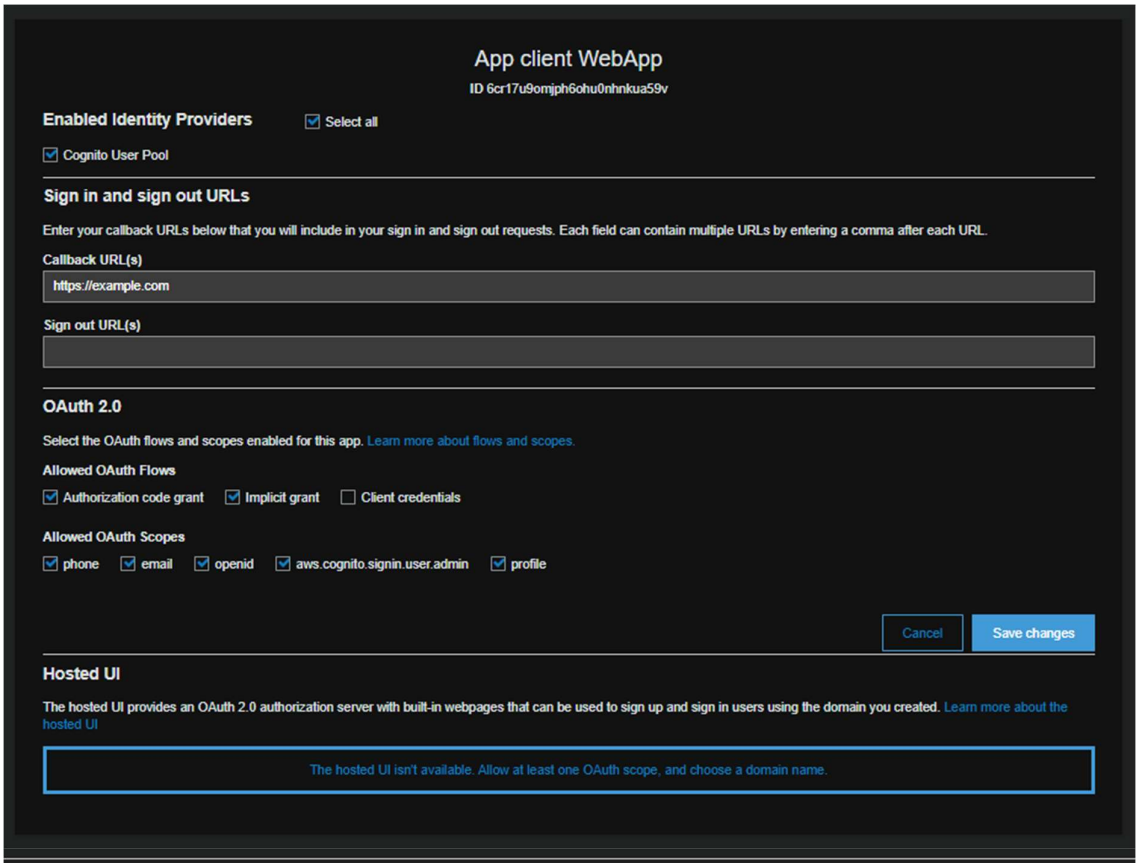
10. 

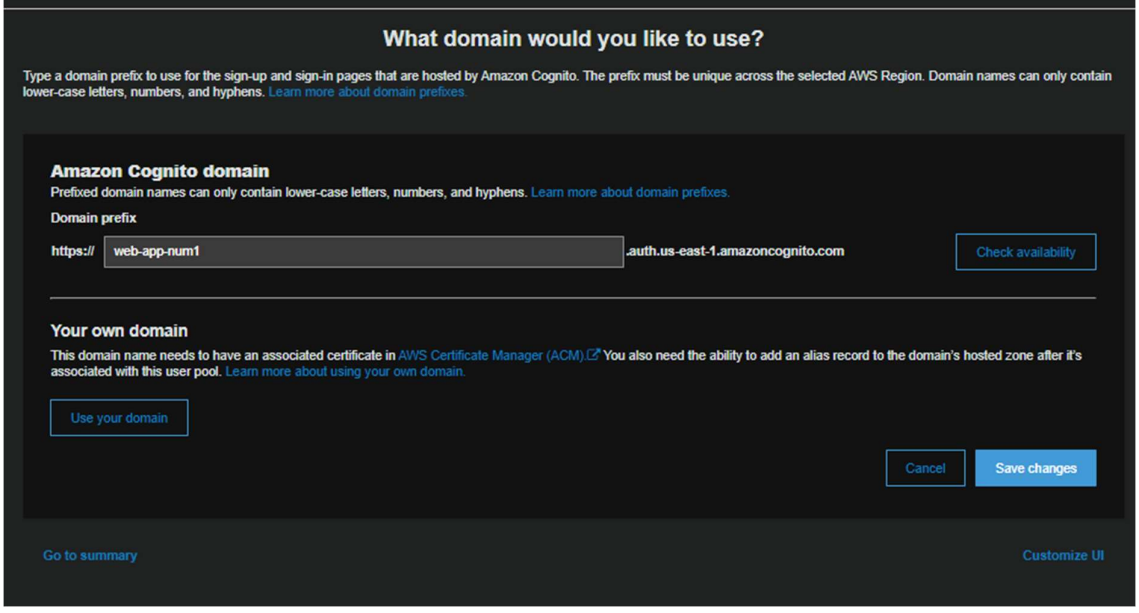
11. 

12. Skip to review.
13. 

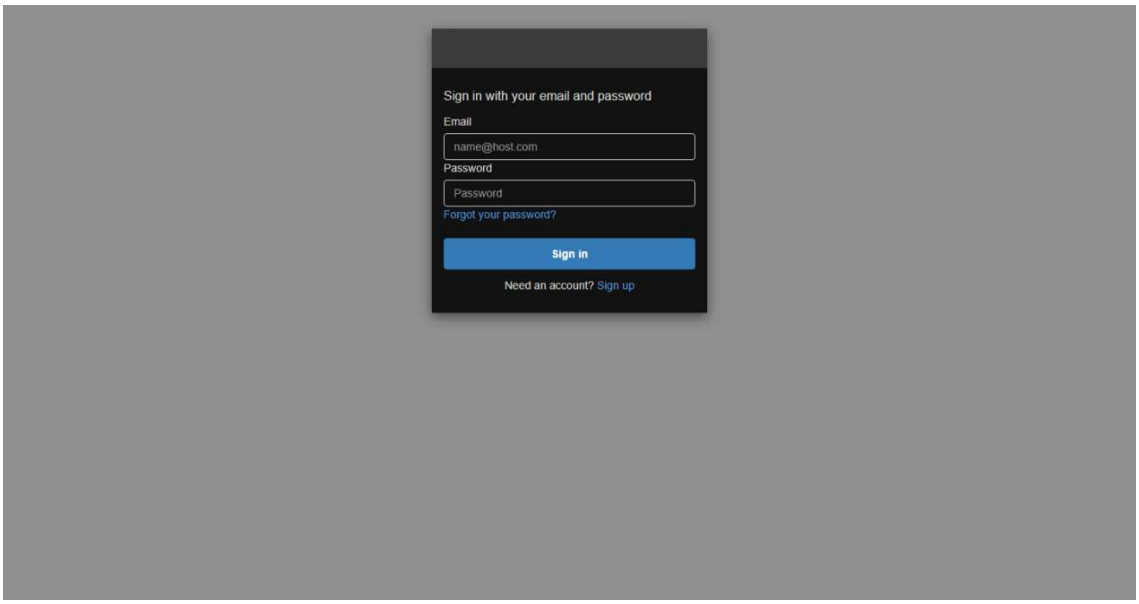
14. 

15. Create app clients, enter the name of the app client, and uncheck the client secret option. 

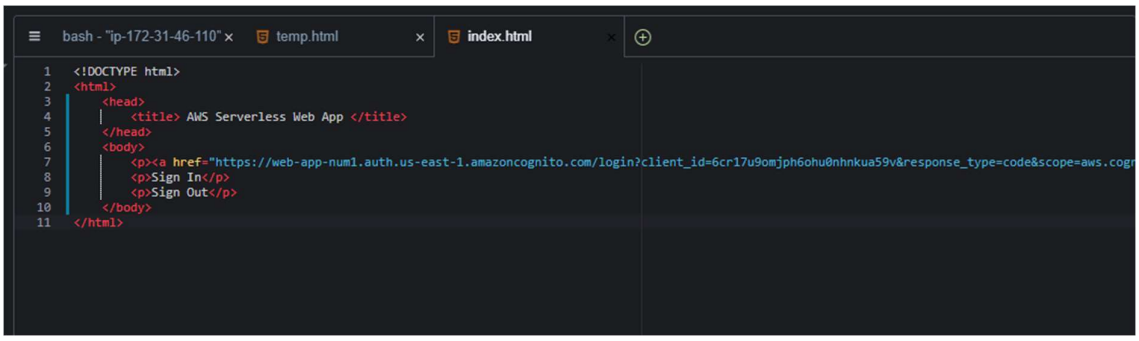
16. 

17. 

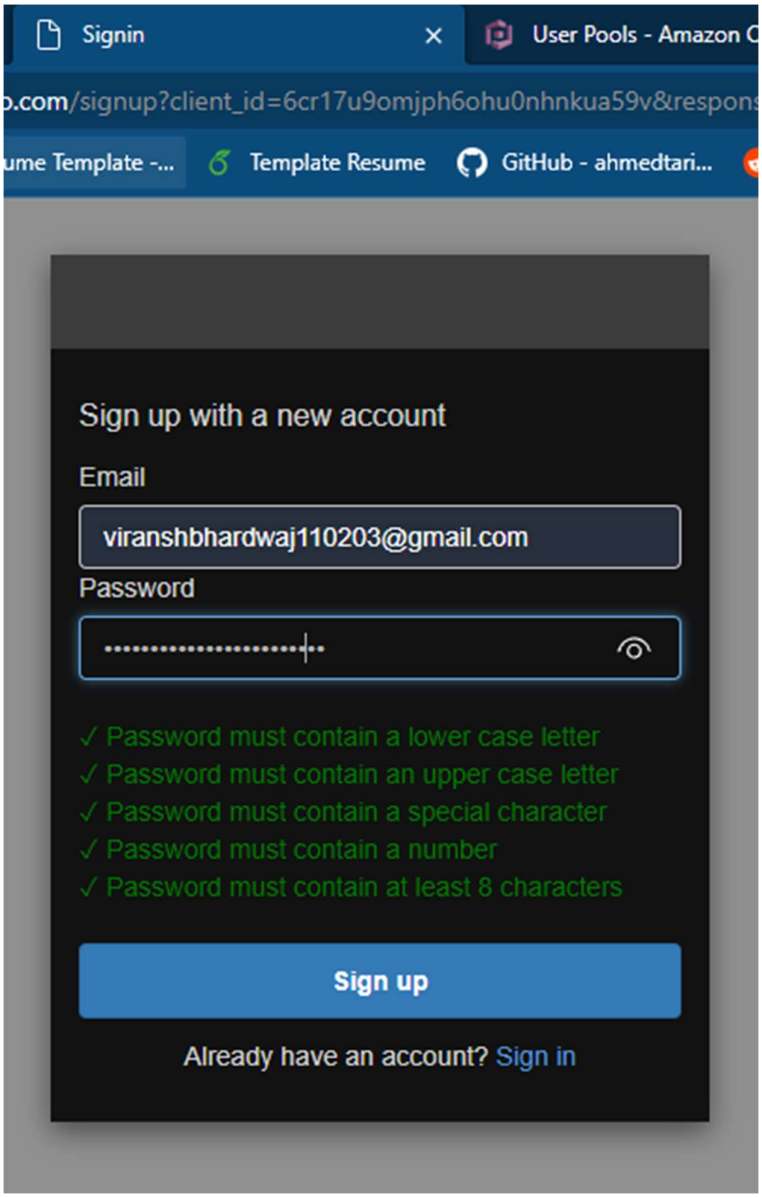
18. Launch the Hosted UI under app client settings.

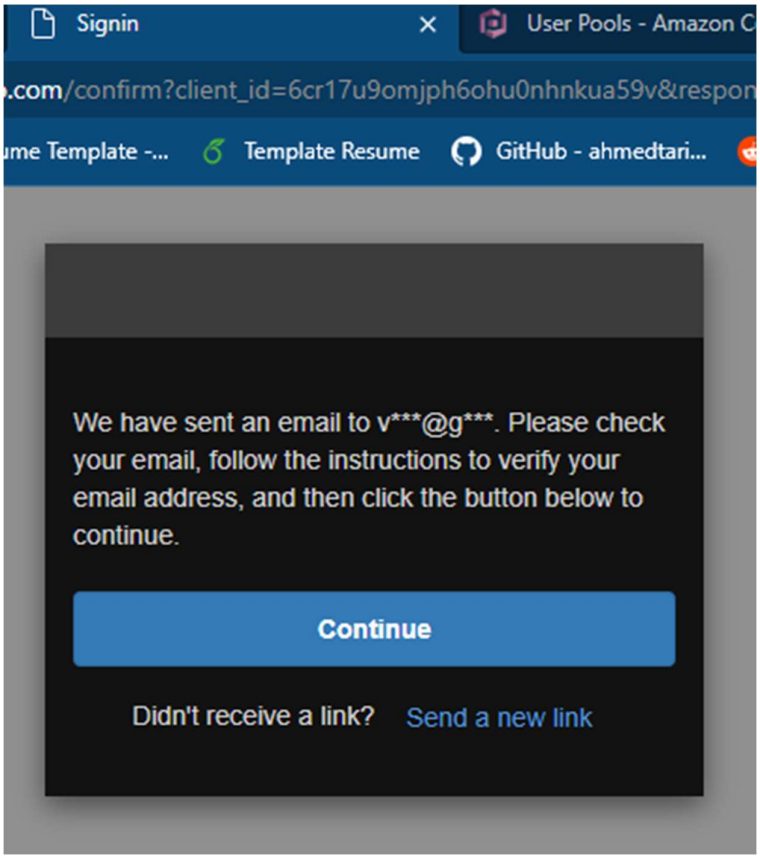
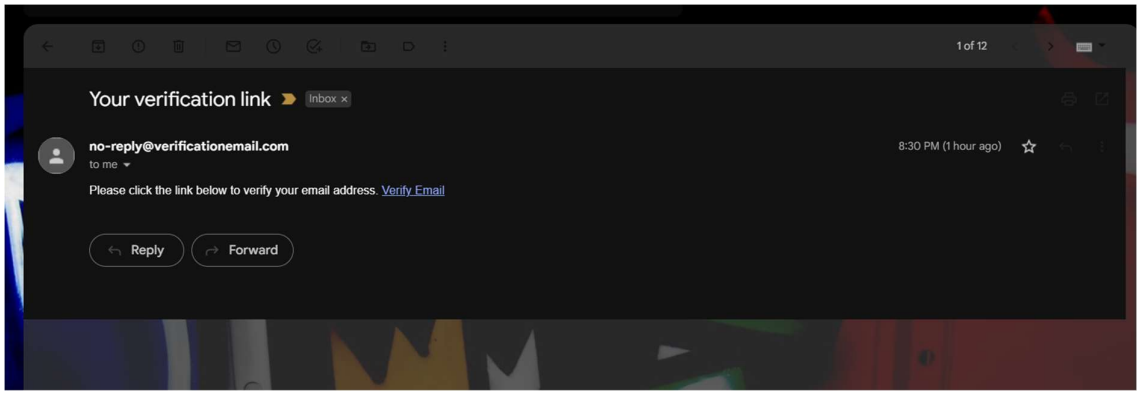
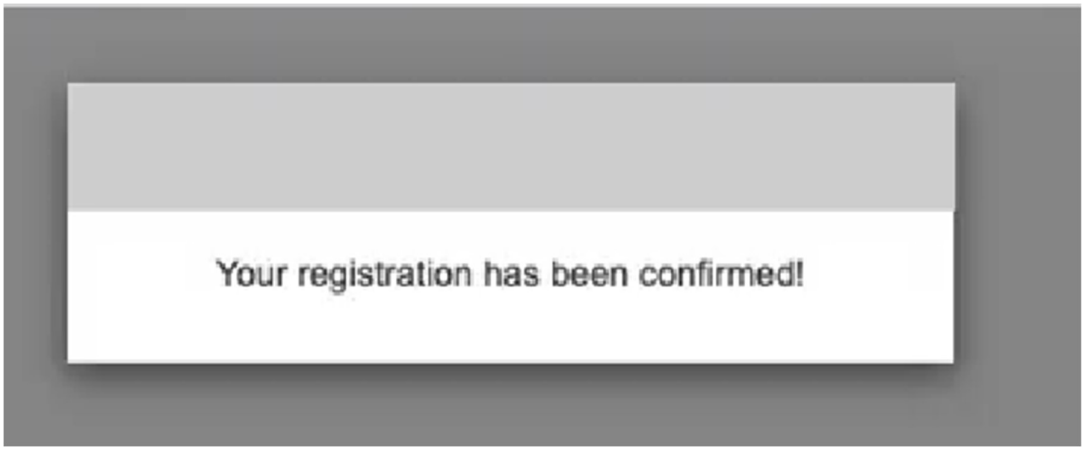
19. 

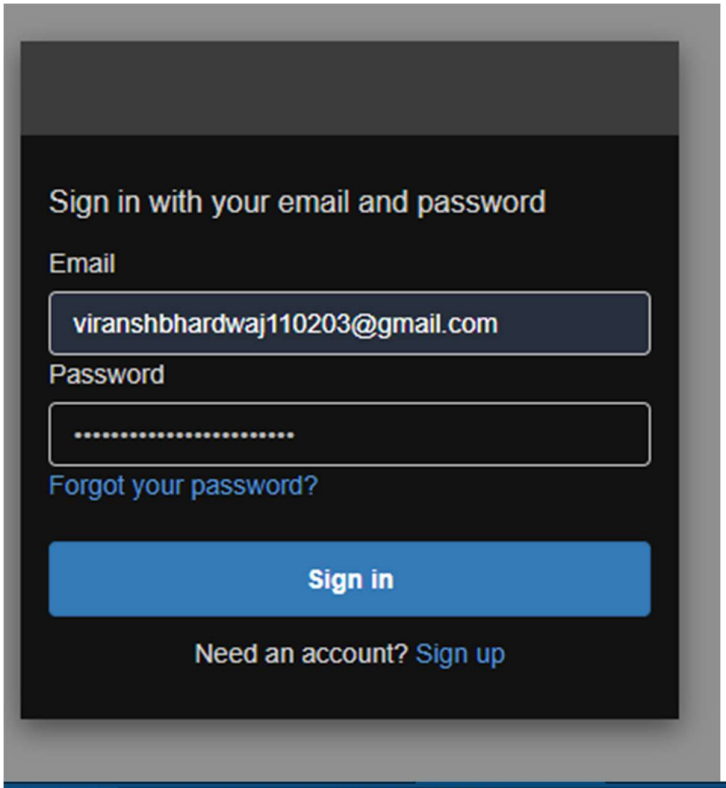
20. Copy the URL of the page obtained and append the url into the *"index.html"* file as shown.

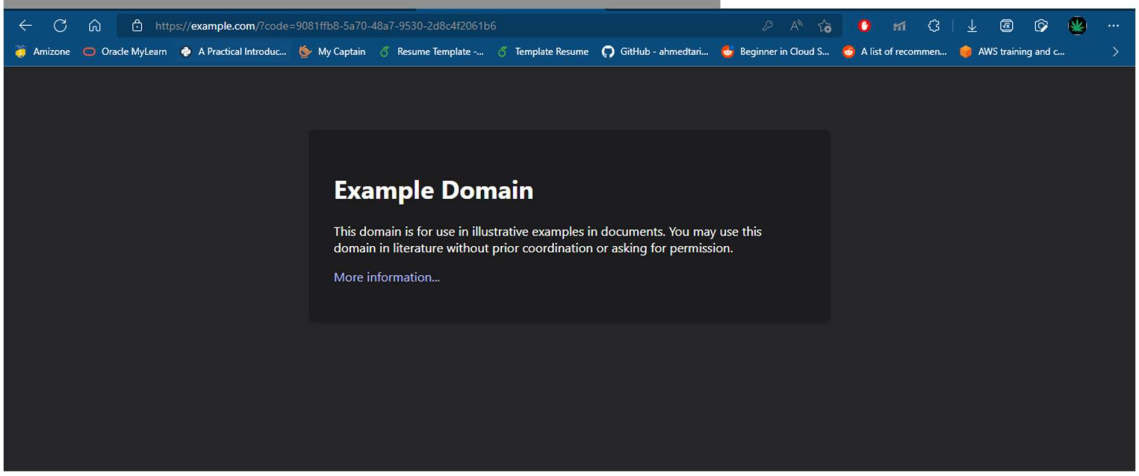


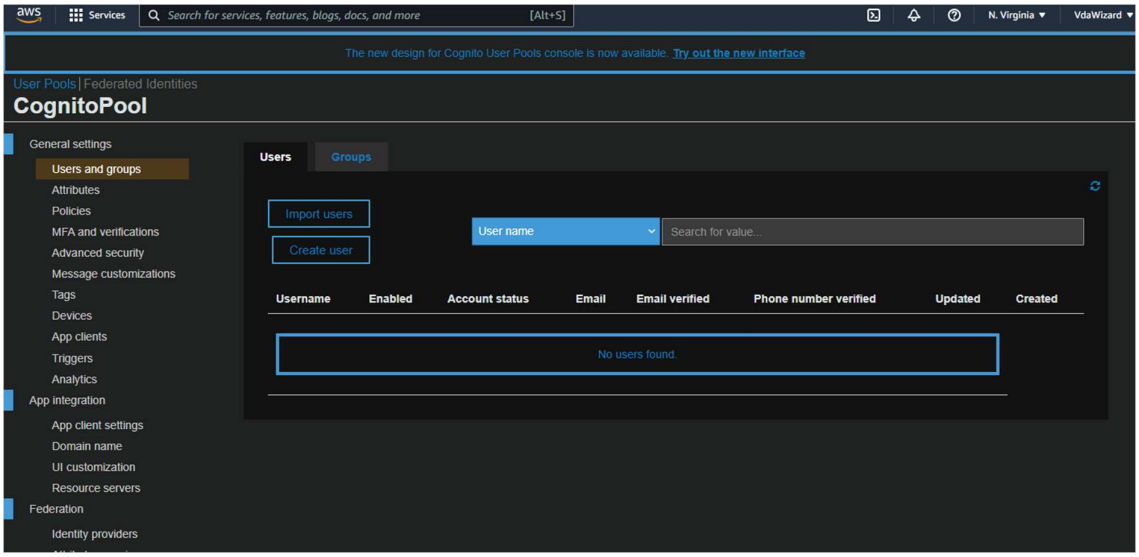
21. Launch the file into a new tab and signup.

22. 
Signin
User Pools - Amazon C
.com/signup?client_id=6cr17u9omjph6ohu0nhnkua59v&respons
ume Template -...
Template Resume
GitHub - ahmedtari...
Sign up with a new account
Email
viranshbhardwaj110203@gmail.com
Password
✓ Password must contain a lower case letter
✓ Password must contain an upper case letter
✓ Password must contain a special character
✓ Password must contain a number
✓ Password must contain at least 8 characters
Sign up
Already have an account? Sign in

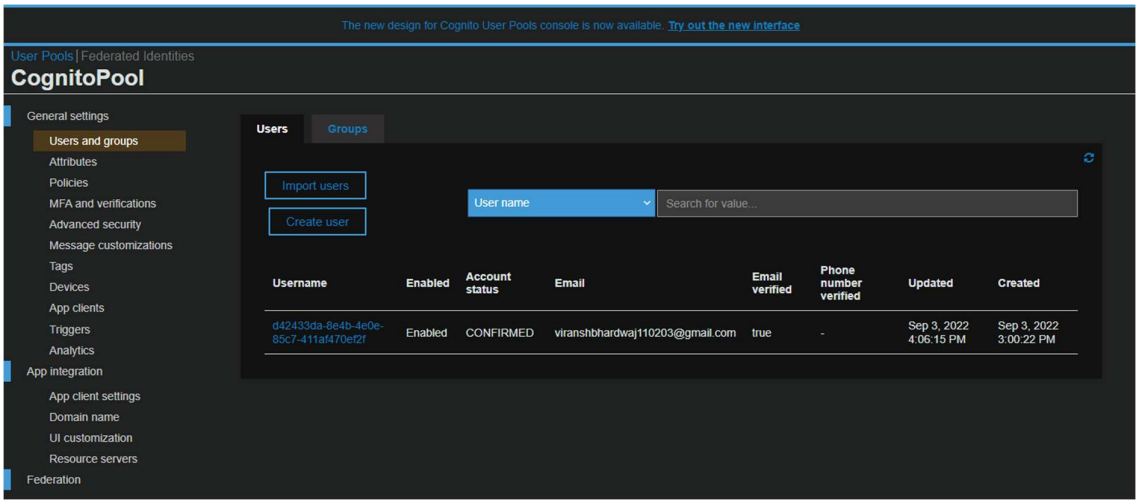
23. 
Signin
User Pools - Amazon C
.com/confirm?client_id=6cr17u9omjph6ohu0nhnkua59v&respon
ume Template -...
Template Resume
GitHub - ahmedtari...
We have sent an email to v***@g***. Please check your email, follow the instructions to verify your email address, and then click the button below to continue.
Continue
Didn't receive a link?
Send a new link
24. 
1 of 12
Your verification link
Inbox ×
no-reply@verificationemail.com
to me
8:30 PM (1 hour ago)
Please click the link below to verify your email address. Verify Email
Reply
Forward
25. 
Your registration has been confirmed!

26. 
Sign in with your email and password
Email
viranshbhardwaj110203@gmail.com
Password
Forgot your password?
Sign in
Need an account? Sign up

27. 
https://example.com/?code=9081ffb8-5a70-48a7-9530-2d8c4f2061b6
Amizone
Oracle MyLearn
A Practical Introduc...
My Captain
Resume Template -...
Template Resume
GitHub - ahmedtari...
Beginner in Cloud S...
A list of recommen...
AWS training and c...
Example Domain
This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.
More information...

28. 

29. Refresh to see the active users.



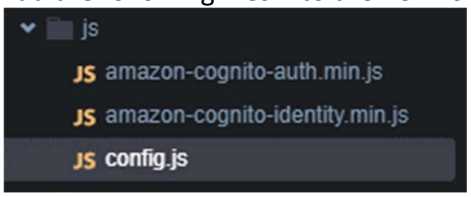
30. Create a new sign-in.html file with the following contents.

```
<!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>
```

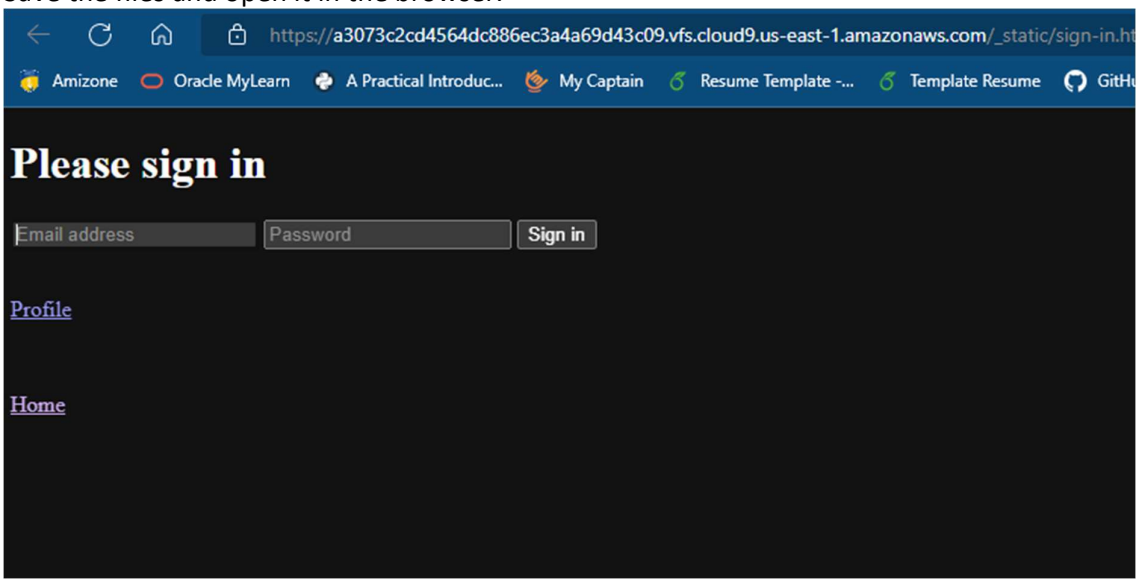
```
    <input type="text" id="inputUsername"  placeholder="Email address"
name="username" required autofocus>
    <input type="password" id="inputPassword"  placeholder="Password"
name="password" required>
    <button type="button" onclick="signInButton()">Sign in</button>
   </form>

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

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

   <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();
                });

            },
            onFailure: function(err) {
              alert(err.message || JSON.stringify(err));
            },
        });
      }
    </script>

  </body>
</html>
```

31. Add the following files into the new folder js.



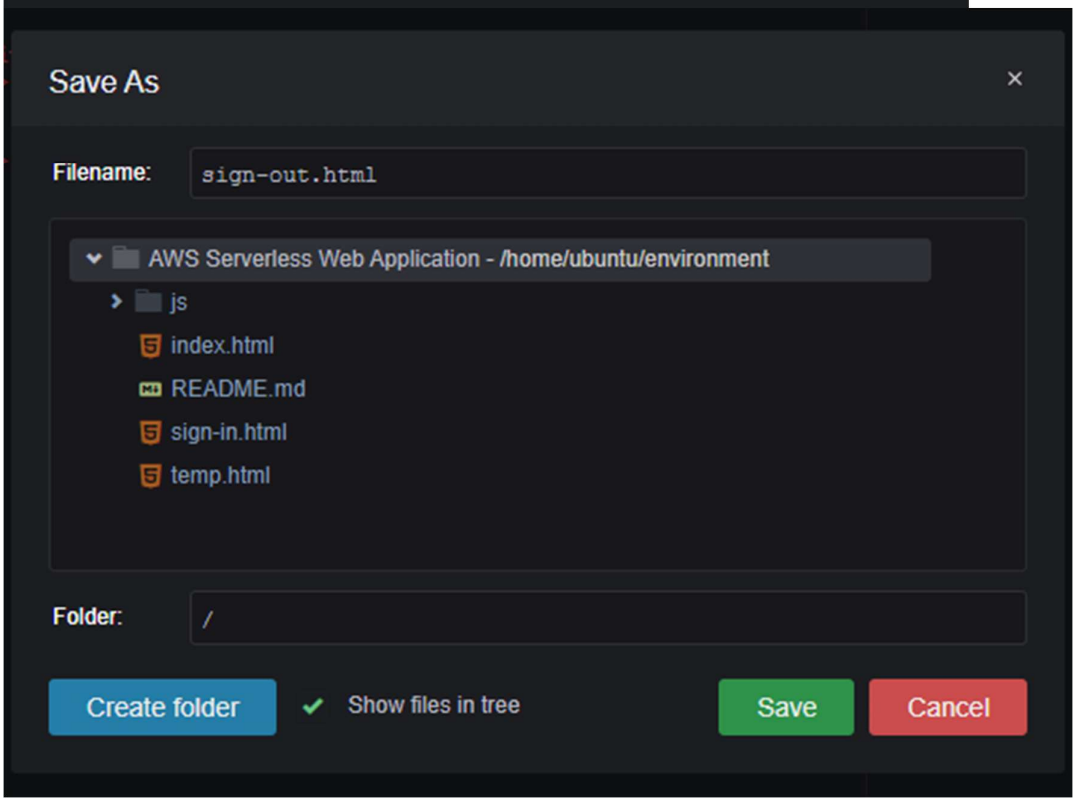
32. The files will be attatched into the upload link.
33. Save the files and open it in the browser.



34. Enter the credentials to see the following screen.

35. To create a sign out option as well, create a new html file '*sign-out.html*', update the index.html file.

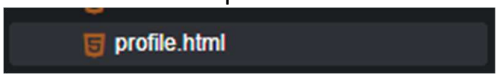
```
<!DOCTYPE html>
<html>
    <head>
        <title> AWS Serverless Web App </title>
    </head>
    <body>
        <p><a href="https://web-app-num1.auth.us-east-1.amazoncognito.c
        <p><a href="./sign-in.html">Sign In</a></p>
        <p><a href="./sign-out.html">Sign Out</a></p>
    </body>
</html>
```



36.
37.

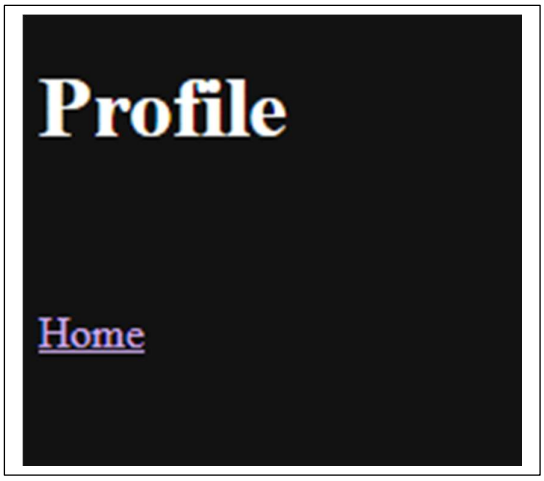
38. 

39. Create a new file ‘profile.html’ with the following contents.




```
<!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,
     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>
```



40. Preview the page and you serverless login page is ready.

NOTE: make sure to add the same email id into the Dynamo DB table so that it may display the data in profile page correctly.

OVERVIEW OF THE USER SIGN IN/OUT/UP/PROFILE

Sign Up: Already signed up as ***viranshbhardwaj110203@gmail.com*** email address.

Sign IN:

SIGN OUT:

Go back to home page and click on sign out.

Sign Out

Successfully signed-out

Home

Project Created By:

VIRANSH BHARDWAJ

GADDALA GREESHMA DEVI

YUKTA RAJESH SHARMA

VIKAS REDDY

SARVESH LATH

Lab files used:

Index.html: https://amityedu96491-my.sharepoint.com/:t:/g/personal/viransh_bhardwaj_s_amity_edu/EbpHuTXcYM1Mg-zHxYEUTjIBqtmmLeoGm-ilE5Q3BX-sXQ?e=24JWAZ

Profile.html: https://amityedu96491-my.sharepoint.com/:t:/g/personal/viransh_bhardwaj_s_amity_edu/EavWN297n5VGjUpMx4reavABlCkVFTf8v9k7Km_TR3XQOw?e=tWeBWG

Sign-in.html: https://amityedu96491-my.sharepoint.com/:t:/g/personal/viransh_bhardwaj_s_amity_edu/EYaejErCRgNKlXRBBdi5ka4BkeyAeU9jEH9hcUxd1NUojw?e=HkhXuB

Temp.html: https://amityedu96491-my.sharepoint.com/:t:/g/personal/viransh_bhardwaj_s_amity_edu/Eb-4gJTugadHrPdwhXTygaYBajbQc_0fI-geZGtH7-QmHg?e=cgbuBf

Js files:

https://amityedu96491-my.sharepoint.com/:f:/g/personal/viransh_bhardwaj_s_amity_edu/EgV0BecNShFBlWfj0mw3V8EBUSy045WxqiG5BvRZq9OD6Q?e=Fhh8lq

PROJECT FOLDER: https://amityedu96491-my.sharepoint.com/:f:/g/personal/viransh_bhardwaj_s_amity_edu/Ek5Dz1HelOlIh3fGjsqLSYwBFFprKpcvCTMADAm5fztoFQ?e=FQWT3m