**Serverless Web App Development Made Easy: A Complete Guide with AWS Amplify, DynamoDB, Lambda and API Gateway**

A diagram of a software flow

Description automatically generated

Get ready to dive into the world of serverless web application development on AWS. In this series, we’ll guide you through the process of creating a dynamic web app that calculates the area of a rectangle based on user-provided length and width values. We’ll leverage the power of AWS Amplify for web hosting, AWS Lambda functions for real-time calculations, DynamoDB for storing and retrieving results, and API Gateway for seamless communication. By the end of this journey, you’ll have the skills to build a responsive and scalable solution that showcases the true potential of serverless architecture. Let’s embark on this development adventure together!

***Prerequisites***

* Have an AWS account. If you don’t have one, sign up [here](https://aws.amazon.com/) and enjoy the benefits of the [Free-Tier Account](https://aws.amazon.com/free/?trk=99f831a2-d162-429a-9a77-a89f6b3bd6cd&sc_channel=ps&ef_id=Cj0KCQjw06-oBhC6ARIsAGuzdw03qBfKD7AA8ejOZPfdI7Pg4XshpRd57Q1p-a8--id9VCM5Cjg78xEaAmJtEALw_wcB%3AG%3As&s_kwcid=AL%214422%213%21645125273267%21e%21%21g%21%21aws+free+tier%2119574556890%21145779847112&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free+Tier+Types=*all&awsf.Free+Tier+Categories=*all)
* Access to the project files: [Amplify Web-app](https://github.com/Kevin-byt/AWS-Projects/tree/main/Amplify)

***Creating the Front-end***

* Use the index.html file from the project files. Or simply open a text editor and copy the following code into an index.html file. Note the part with “YOUR API URL” as we will be filling this part with the API URL later

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="UTF-8">  
 <title>Rectangle</title>  
 <!-- Styling for the client UI -->  
 <style>  
 h1 {  
 color: #FFFFFF;  
 font-family: system-ui;  
 margin-left: 20px;  
 }  
 body {  
 background-color: #222629;  
 }  
 label {  
 color: #86C232;  
 font-family: system-ui;  
 font-size: 20px;  
 margin-left: 20px;  
 margin-top: 20px;  
 }  
 button {  
 background-color: #86C232;  
 border-color: #86C232;  
 color: #FFFFFF;  
 font-family: system-ui;  
 font-size: 20px;  
 font-weight: bold;  
 margin-left: 30px;  
 margin-top: 20px;  
 width: 140px;  
 }  
 input {  
 color: #222629;  
 font-family: system-ui;  
 font-size: 20px;  
 margin-left: 10px;  
 margin-top: 20px;  
 width: 100px;  
 }  
 </style>  
 <script>  
 // callAPI function that takes the length and width numbers as parameters  
 var callAPI = (length,width)=>{  
 // instantiate a headers object  
 var myHeaders = new Headers();  
 // add content type header to object  
 myHeaders.append("Content-Type", "application/json");  
 // using built in JSON utility package turn object to string and store in a variable  
 var raw = JSON.stringify({"length":length,"width":width});  
 // create a JSON object with parameters for API call and store in a variable  
 var requestOptions = {  
 method: 'POST',  
 headers: myHeaders,  
 body: raw,  
 redirect: 'follow'  
 };  
 // make API call with parameters and use promises to get response  
 fetch("YOUR API URL", requestOptions)  
 .then(response => response.text())  
 .then(result => alert(JSON.parse(result).body))  
 .catch(error => console.log('error', error));  
 }  
 </script>  
</head>  
<body>  
 <h1>AREA OF A RECTANGLE!</h1>  
 <form>  
 <label>Length:</label>  
 <input type="text" id="length">  
 <label>Width:</label>  
 <input type="text" id="width">  
 <!-- set button onClick method to call function we defined passing input values as parameters -->  
 <button type="button" onclick="callAPI(document.getElementById('length').value,document.getElementById('width').value)">CALCULATE</button>  
 </form>  
</body>  
</html>

2. The file should look like this when opened on a browser. It gives spaces to input the length and width of a rectangle and a ‘Calculate’ button

A screenshot of a computer

Description automatically generated

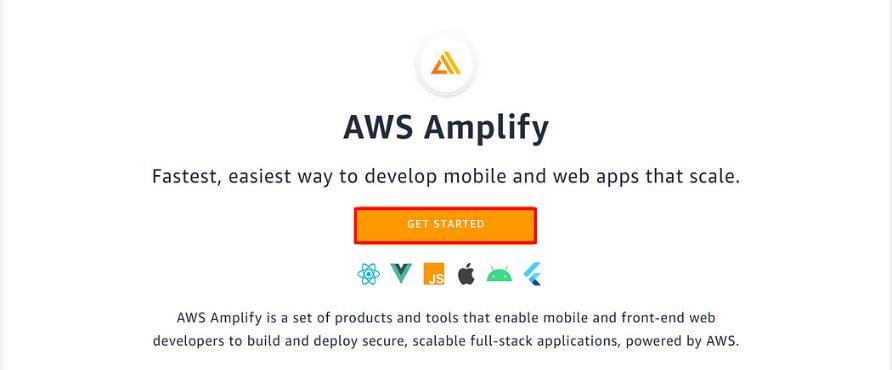
***Hosting the App on AWS Amplify***

* On your AWS console search box, search for ‘Amplify’ and click on the first option that appears

A screenshot of a computer

Description automatically generated

2. Click on ‘GET STARTED’



3. Select ‘Get Started’ on the Amplify Hosting side

A screenshot of a web page

Description automatically generated

4. select the source for your app files. They can be in a remote repository or local. We will use ‘Deploy without Git provider’ since our files are local. We also need to use a compressed folder with our files. Click on ‘Continue’

A screenshot of a computer

Description automatically generated

5. Give the app a name, an environment name, choose the method as ‘Drag and drop’ and selct the index.zip file (zip all the app files. In this case, it is only the index.html file). Click on ‘Save and deploy’

A screenshot of a computer

Description automatically generated

6. Once the deployment is complete, click on the Domain to access your app

A screenshot of a computer

Description automatically generated

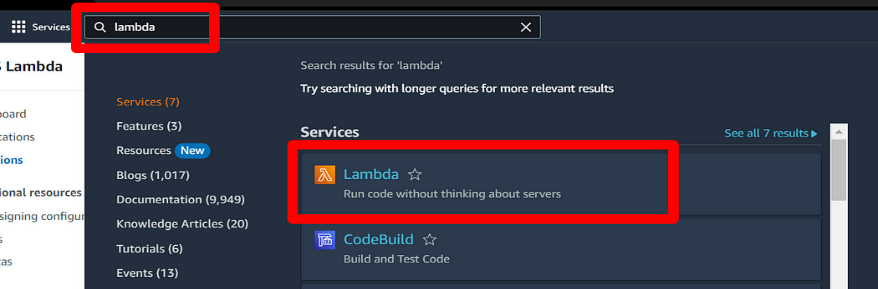
7. The app opens on the browser. (You might need to refresh the deployment page on Amplify. Maybe it’s a bug or something 😉)

A screenshot of a computer

Description automatically generated

***Creating a Lambda Function to do the Math***

1.On the AWS console search bar, type ‘Lambda’ and select the Lambda service



2. Click on ‘Create function’

A screenshot of a computer

Description automatically generated

3. Give the Function name, The Runtime(Latest Python), then scroll down and click on ‘Create function’

A screenshot of a computer

Description automatically generated

4. Copy the following Lambda function onto your lambda\_function.py file. Please note the DynamoDB name. We will be using this name later as we create the DB.

# import the JSON utility package  
import json  
  
# import the AWS SDK (for Python the package name is boto3)  
import boto3  
  
# import two packages to help us with dates and date formatting  
from time import gmtime, strftime  
  
# create a DynamoDB object using the AWS SDK  
dynamodb = boto3.resource('dynamodb')  
  
# use the DynamoDB object to select our table  
table = dynamodb.Table('AreaDatabase')  
  
# store the current time in a human readable format in a variable  
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())  
  
# define the handler function that the Lambda service will use an entry point  
def lambda\_handler(event, context):  
  
# extract the two numbers from the Lambda service's event object  
 Area = int(event['length']) \* int(event['width'])  
  
# write result and time to the DynamoDB table using the object we instantiated and save response in a variable  
 response = table.put\_item(  
 Item={  
 'ID': str(Area),  
 'LatestGreetingTime':now  
 })  
  
# return a properly formatted JSON object  
 return {  
 'statusCode': 200,  
 'body': json.dumps('Your result is ' + str(Area))  
 }

5. Click on ‘Deploy’

A screenshot of a computer

Description automatically generated

***Create an API Gateway***

On the AWS services search box, enter ‘API’ and select ‘API Gateway’ that appears

A screenshot of a computer

Description automatically generated

2. In the list for ‘Choose an API type’, select ‘Build’ for ‘REST API’

A screenshot of a computer

Description automatically generated

3. Choose the ‘REST’ protocol for the API, select ‘New API’ under ‘Create new API’ and give the API a name, then click on ‘Create API’

A screenshot of a computer

Description automatically generated

4. On the page that appears, select ‘Resources’ on the Left Panel, On the ‘Actions’ drop-down, select ‘Create method’. Select ‘POST’ on the drop down that appears then click on the ✔. Select ‘Lambda Function’ as the Integration type and type the name of the lambda function in the ‘Lambda Function’ box. Click on ‘Save’

A screenshot of a computer

Description automatically generated

5. On the dialog box that appears to Add Permission to Lambda Function, click ‘OK’

A screenshot of a computer

Description automatically generated

6. Select ‘POST’. On the ‘Actions’ drop down, click on ‘Enable CORS/ then click on ‘Enable CORS and replace existing CORS headers’ on the bottom right

A screenshot of a computer

Description automatically generated

7. On the ‘Confirm method changes’ box that appears, click on ‘Yes, replace existing values’

A screenshot of a computer

Description automatically generated

8. Once all the checks are complete, click on ‘Actions, then, ‘Deploy API’

A screenshot of a computer

Description automatically generated

9. Give the ‘Stage name’, then click ‘Deploy’

A screenshot of a computer

Description automatically generated

10. The Invoke URL is what you replace “YOUR API URL” with on the index.html file. Insert the URL, regenerate the index.zip and reupload to Amplify

A screenshot of a computer

Description automatically generated

Invoke URL: https://yuavndwnn4.execute-api.us-east-1.amazonaws.com/dev

***Setting up a Database on DynamoDB to store results***

On the services search box, search for ‘DynamoDB’ and select the DynamoDB service

A screenshot of a computer

Description automatically generated

2. Click on ‘Create table’

A screenshot of a computer

Description automatically generated

3. Give the table a name, for ‘Partition key’ input ‘ID’. Leave the rest as default, scroll to the bottom and click on ‘Create table’

A screenshot of a computer

Description automatically generated

4. Select the table name. Under the overview tab, expand ‘Additional info’, then take note of the ARN

A screenshot of a computer

Description automatically generated

arn:aws:dynamodb:us-east-1:494225556983:table/Area\_table

5. Let’s add permissions to our Lambda function to access DynamoDB. On the Lambda function window, select the ‘Configuration’ tab, then ‘Permissions’ on the left side panel and select the Role name.

A screenshot of a computer

Description automatically generated

6. A new tab opens in IAM and we can add permissions to the role. Click on ‘Add permissions’, then ‘Create inline policy’

A screenshot of a computer

Description automatically generated

7. Select the JSON Tab and copy the following policy. Replace “YOUR-TABLE-ARN” with the ARN of your table that we copied in step 4, then click ‘Next’ at the bottom

A screenshot of a computer

Description automatically generated

{  
"Version": "2012-10-17",  
"Statement": [  
 {  
 "Sid": "VisualEditor0",  
 "Effect": "Allow",  
 "Action": [  
 "dynamodb:PutItem",  
 "dynamodb:DeleteItem",  
 "dynamodb:GetItem",  
 "dynamodb:Scan",  
 "dynamodb:Query",  
 "dynamodb:UpdateItem"  
 ],  
 "Resource": "YOUR-TABLE-ARN"  
 }  
 ]  
}

8. On the ‘Review and create’ page, give the policy a name the click on ‘Create policy’ at the bottom of the page

A screenshot of a computer

Description automatically generated

9. Now the Lambda function has permissions to write to the DB

***Testing***

Now that we are done, let’s see what we have. Open the AWS Amplify domain. It should open our app.

A screenshot of a computer

Description automatically generated

2. Input values for the Length and Width and click on “Calculate”. The solution should pop up on the screen. (Returned in the browserthrough API Gateway)

A screenshot of a computer

Description automatically generated

3. Yaaaay!!!!! And we are successful

***Delete your Resources***

Remember to delete your resources to avoid unnecessary charges:

Delete the Amplify App

Delete the DynamoDB Table

Delete the Lambda function

Delete the API Gateway

**Conclusion**

In this comprehensive guide, we’ve embarked on an exciting journey into the realm of serverless web application development on AWS. We’ve built a dynamic web app that calculates the area of a rectangle based on user-provided length and width values. Leveraging the power of AWS Amplify for web hosting, AWS Lambda functions for real-time calculations, DynamoDB for result storage, and API Gateway for seamless communication, we’ve demonstrated the incredible potential of serverless architecture.