

# **AWS Practice Project: Deploy a Cost-Free Serverless Web Application**

## Project Overview

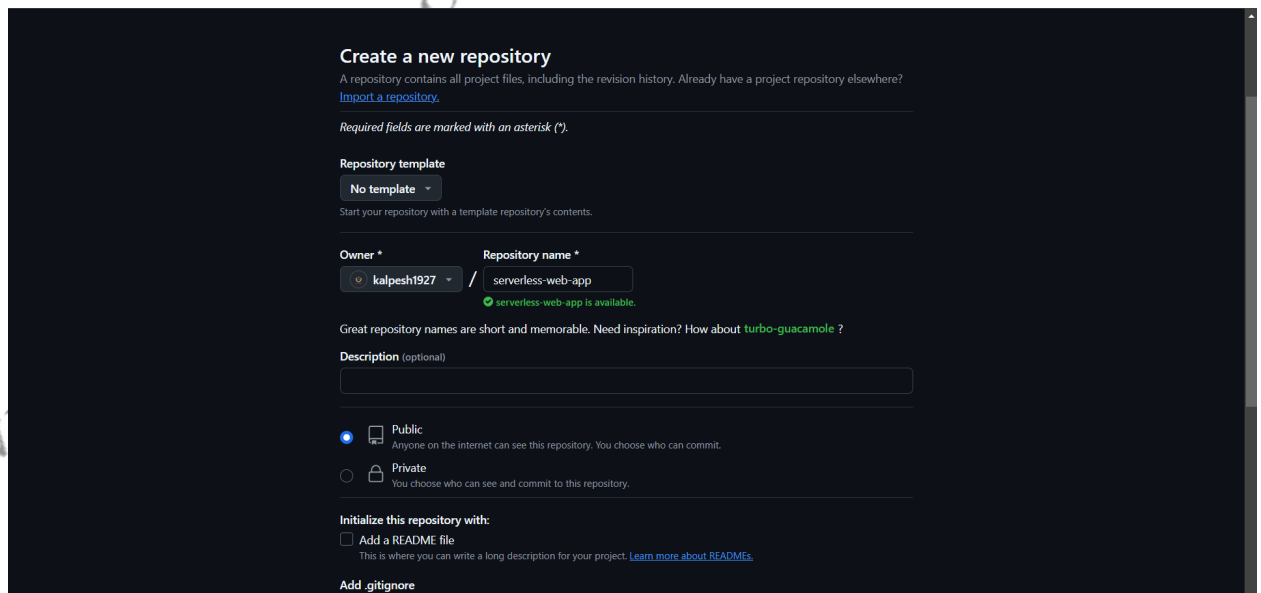
This project involves creating a simple serverless web application where users can submit feedback. The application will use **GitHub Pages** for free website hosting, **AWS Lambda (under Free Tier)** for backend logic, **Amazon API Gateway (within Free Tier limits)** for API management, and **DynamoDB Free Tier** for storing user feedback.

## AWS Services Used (All Within Free Tier)

- **GitHub Pages** - Free hosting for the static website (Alternative to S3)
- **AWS Lambda** - Free Tier allows 1M requests/month
- **Amazon API Gateway** - Free Tier allows 1M API calls/month
- **Amazon DynamoDB** - Free Tier allows 25GB storage and 25 WCU/RCU
- **AWS IAM** - Manage permissions and security

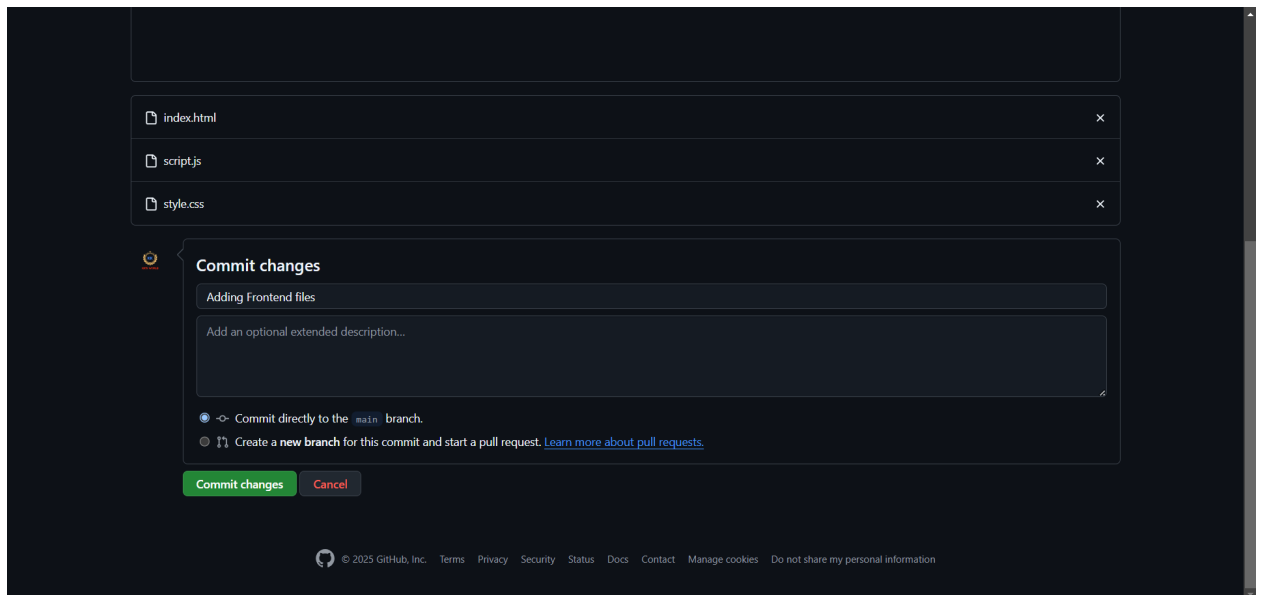
## Step 1: Host the Static Website on GitHub Pages (Free Alternative to S3)

1. Create a GitHub Repository named serverless-web-app.

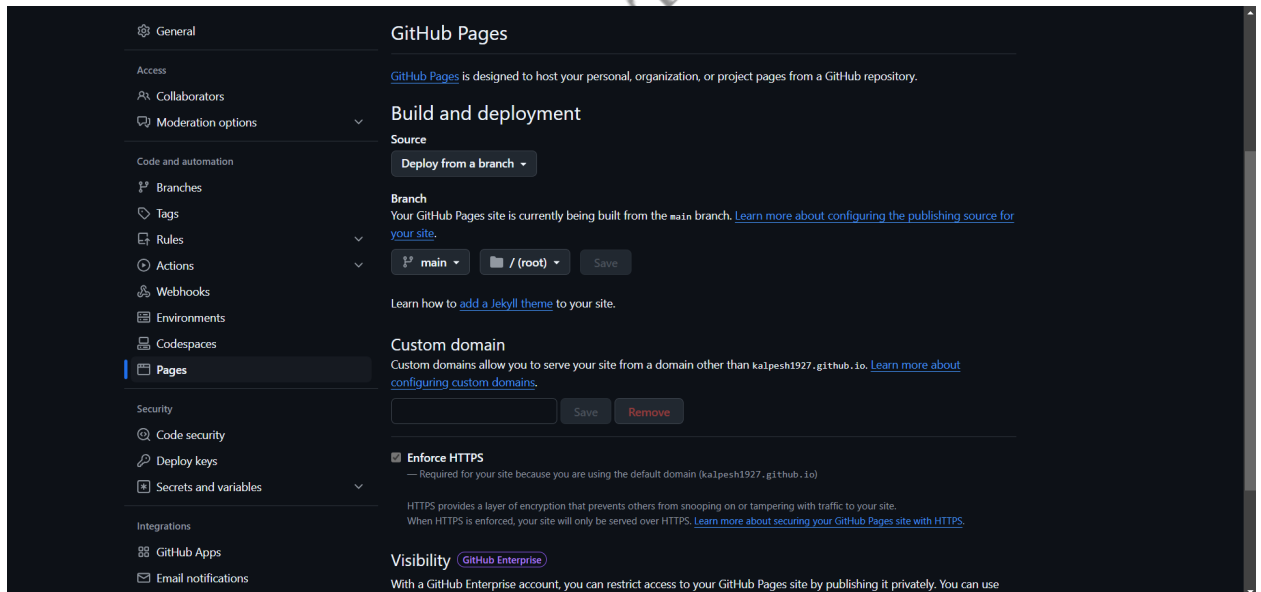


The screenshot shows the GitHub 'Create a new repository' page. The form includes fields for 'Repository template' (set to 'No template'), 'Owner' (kalpesh1927), and 'Repository name' (serverless-web-app). A green checkmark indicates that 'serverless-web-app' is available. There is a text input field for 'Description (optional)'. Under 'Visibility', the 'Public' radio button is selected. At the bottom, there is a checkbox for 'Add a README file' and a link to 'Learn more about READMEs'. The 'Add .gitignore' link is also visible.

2. Upload the frontend files (index.html, style.css, script.js).



3. Go to **Settings** → **Pages** and select the main branch for deployment.



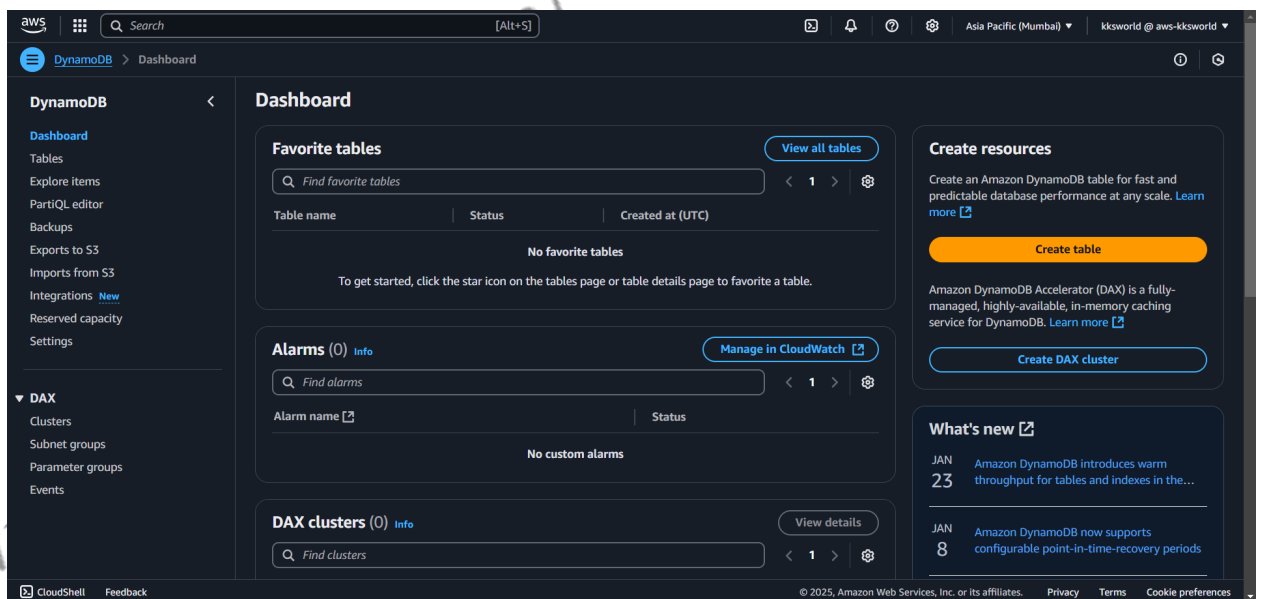
4. GitHub will provide a free website URL (e.g., <https://kalpesh1927.github.io/serverless-web-app/>).

### Submit Your Feedback

Submit

## Step 2: Set Up a DynamoDB Table (Free Tier Limits)

1. Go to the AWS DynamoDB Console and create a new table.



2. Table name: UserFeedback

**Create table**

**Table details** [info](#)  
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

**Table name**  
This will be used to identify your table.

UserFeedback

Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).

3. Set **Partition Key** : feedback\_id (String).

**Partition key**  
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

feedback\_id

String

1 to 255 characters and case sensitive.

4. Click "Create Table" and note the table name.

**AWS** | Search | [Alt+S] | Asia Pacific (Mumbai) | kksworld @ aws-kksworld

**DynamoDB** > Tables

**Dashboard**  
**Tables**  
Explore items  
 PartiQL editor  
 Backups  
 Exports to S3  
 Imports from S3  
 Integrations [New](#)  
 Reserved capacity  
 Settings

**DAX**  
Clusters  
Subnet groups  
Parameter groups  
Events

**The UserFeedback table was created successfully.**

**Tables (1)** [info](#)

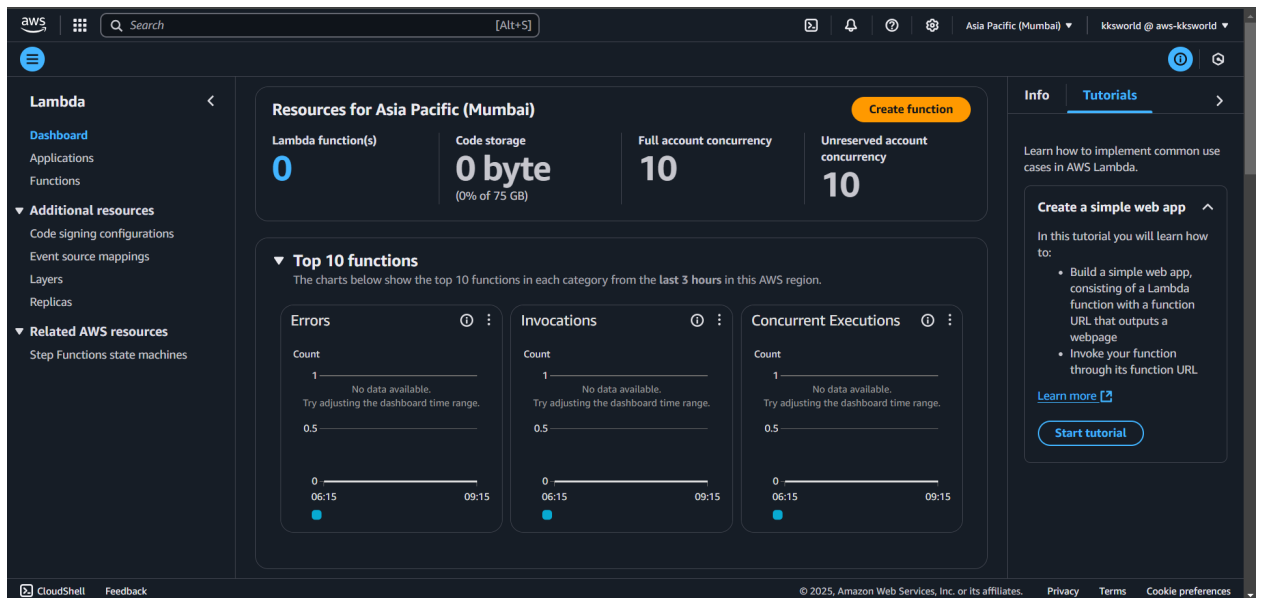
Find tables | Any tag key | Any tag value | < 1 > | Settings

<input type="checkbox"/>	Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode
<input type="checkbox"/>	UserFeedback	Active	feedback_id (S)	-	0	0	Off	☆	On-demand

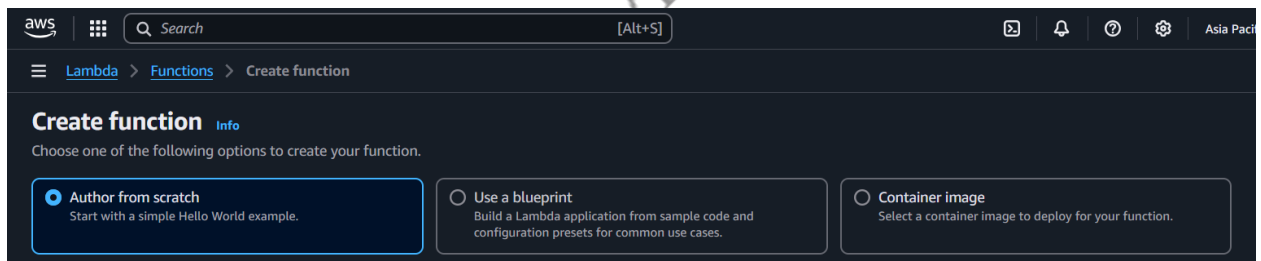
CloudShell | Feedback | © 2025, Amazon Web Services, Inc. or its affiliates. | Privacy | Terms | Cookie preferences

## Step 3: Create an AWS Lambda Function (Under Free Tier)

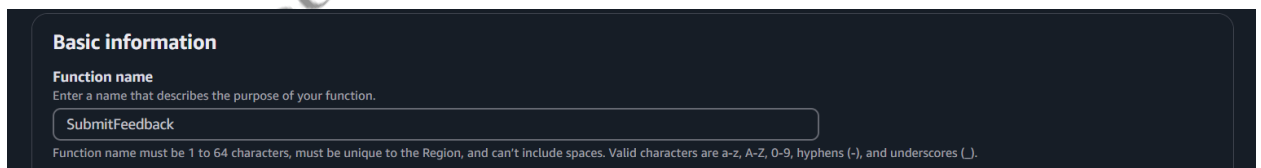
1. Go to the AWS Lambda Console and create a new function.



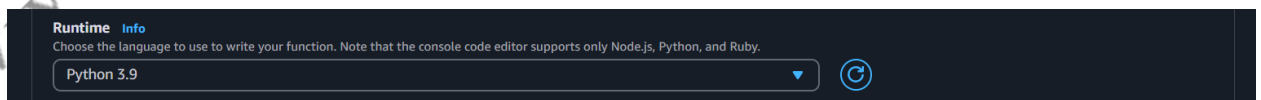
2. Choose **Author from scratch**.



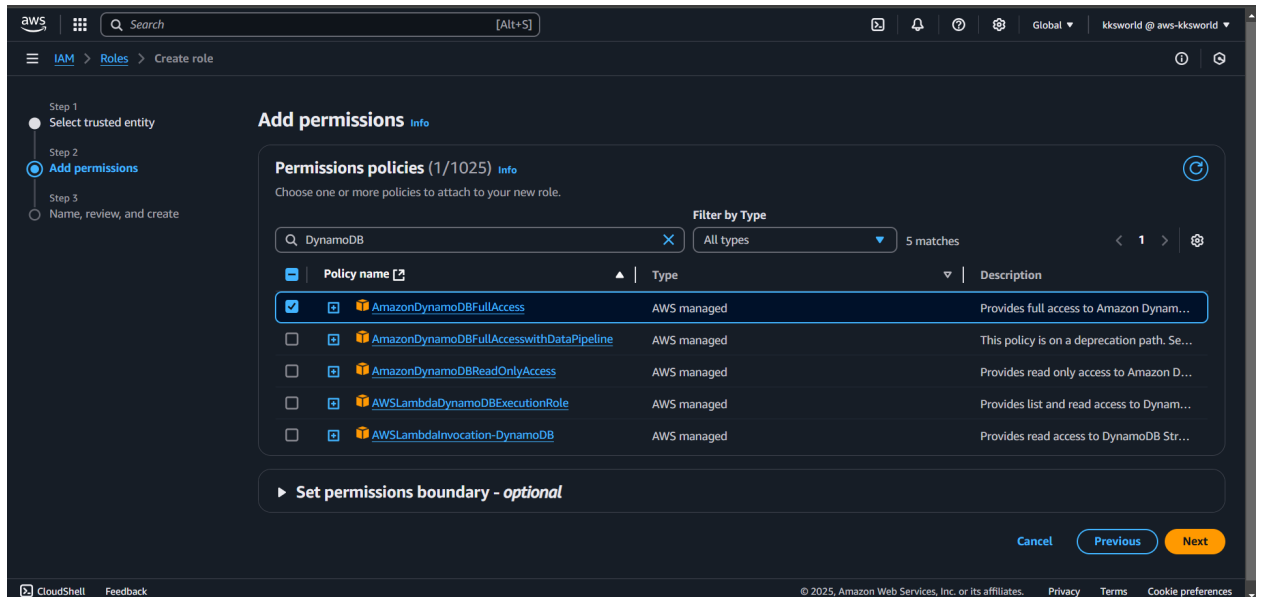
3. Function Name: SubmitFeedback



4. Runtime: **Python 3.9** or **Node.js 18**.



5. Create a new IAM role with DynamoDB Read/Write Access permissions.



- If "AmazonDynamoDBReadWriteAccess " is unavailable, search for AmazonDynamoDBFullAccess instead.
- This grants full control over all DynamoDB tables, which is not ideal for security.

6. Add the following code to handle feedback submission:

#### Python Code Example:

```
import json
import boto3
import uuid
from datetime import datetime

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

def lambda_handler(event, context):
    data = json.loads(event['body'])
    feedback_id = str(uuid.uuid4())
```

```
timestamp = datetime.utcnow().isoformat()
```

```
table.put_item(
```

```
    Item={
```

```
        'feedback_id': feedback_id,
```

```
        'user_name': data['user_name'],
```

```
        'email': data['email'],
```

```
        'message': data['message'],
```

```
        'timestamp': timestamp
```

```
    }
```

```
)
```

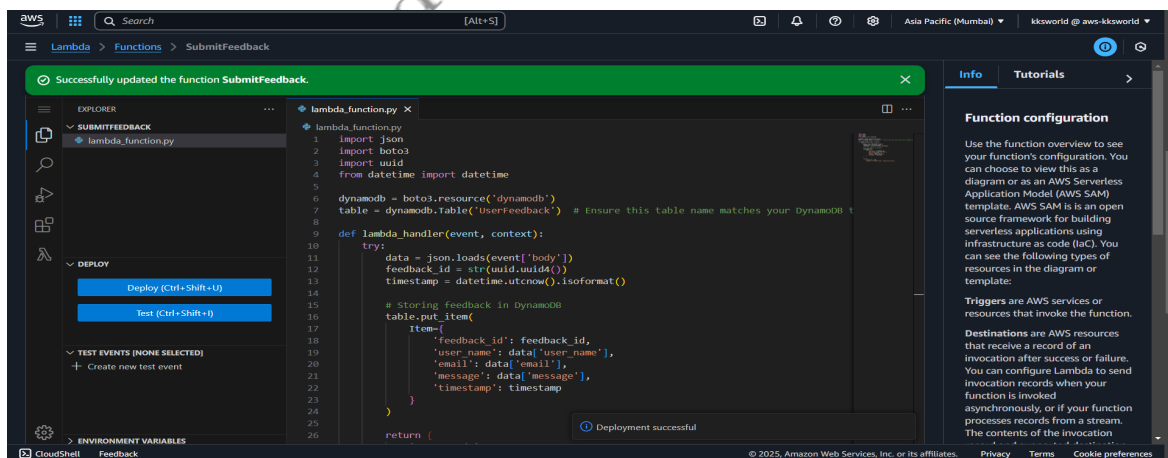
```
return {
```

```
    'statusCode': 200,
```

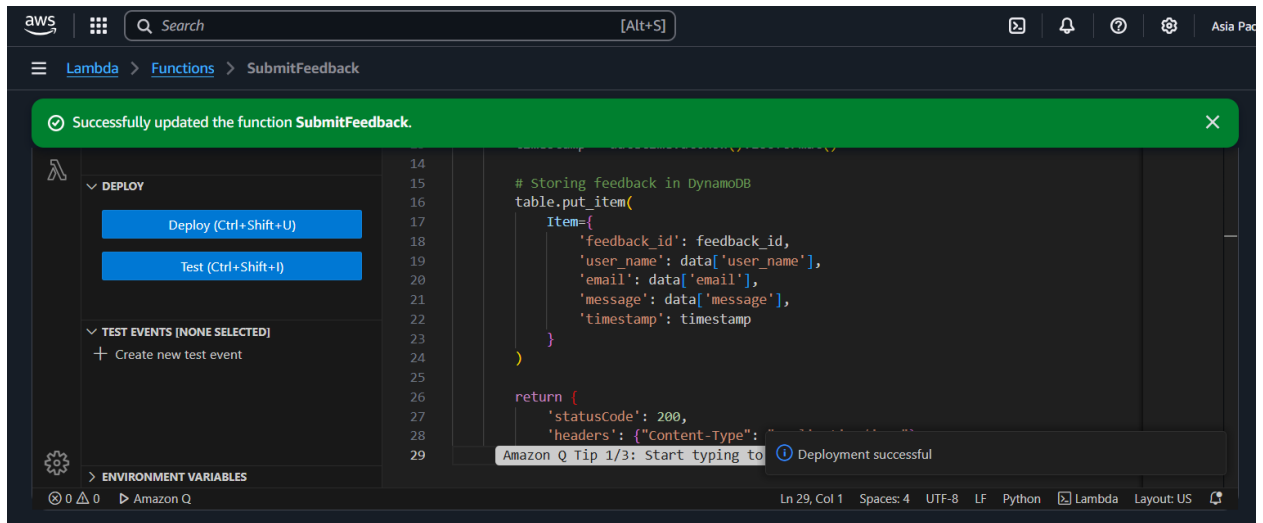
```
    'headers': {"Content-Type": "application/json"},
```

```
    'body': json.dumps({'message': 'Feedback submitted successfully'})
```

```
}
```

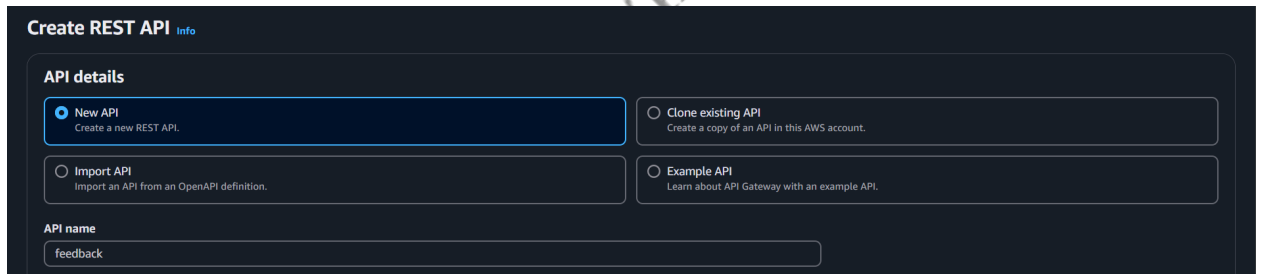


## 7. Deploy the function and note the ARN.



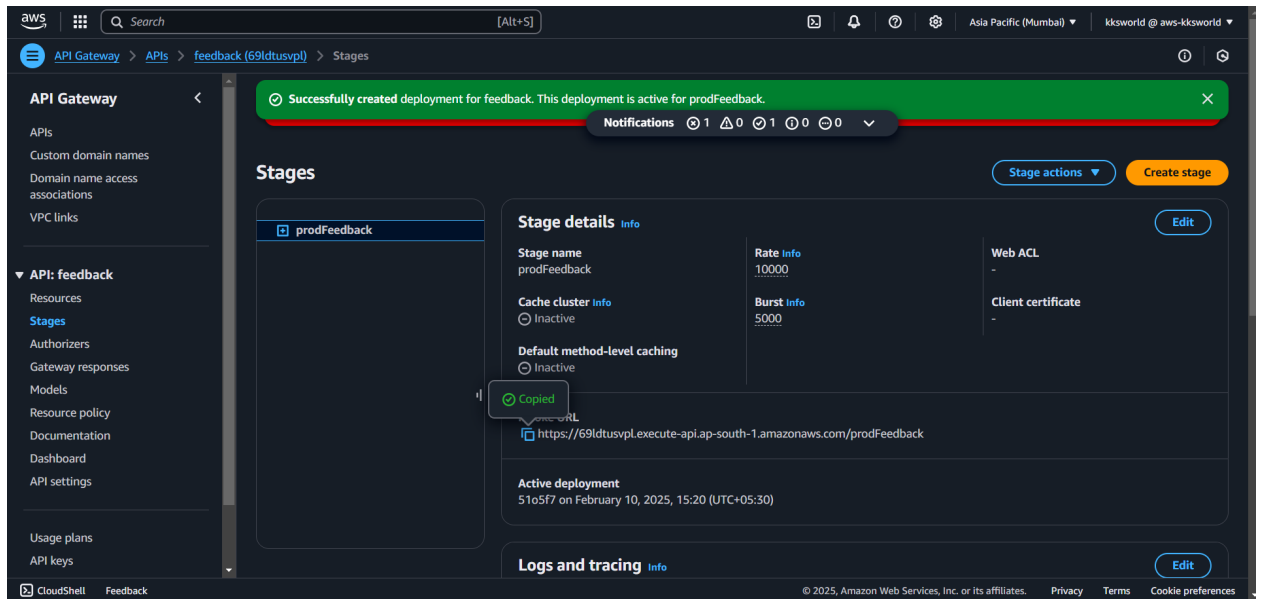
#### Step 4: Configure API Gateway (Free Tier Limits)

1. Go to the API Gateway Console and create a new REST API.
2. Create a **Resource** : /feedback.



3. Add a **POST Method** and integrate it with your Lambda function.
4. Deploy the API and note the Invoke URL.





## Step 5: Connect the Frontend to API Gateway

1. Update the script.js file to use the API Gateway **Invoke URL** :

```
async function submitFeedback() {
```

```
  const data = {
```

```
    user_name: document.getElementById("name").value,
```

```
    email: document.getElementById("email").value,
```

```
    message: document.getElementById("message").value
```

```
  };
```

```
  const response = await fetch("YOUR_API_GATEWAY_URL/feedback", {
```

```
    method: "POST",
```

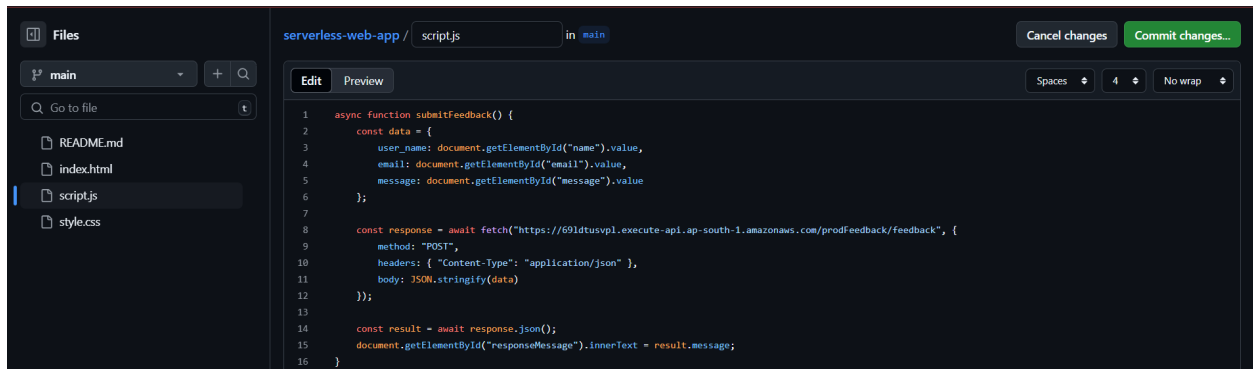
```
    headers: { "Content-Type": "application/json" },
```

```
    body: JSON.stringify(data)
```

```
  });
```

```
  const result = await response.json();
```

```
  alert(result.message);}
```

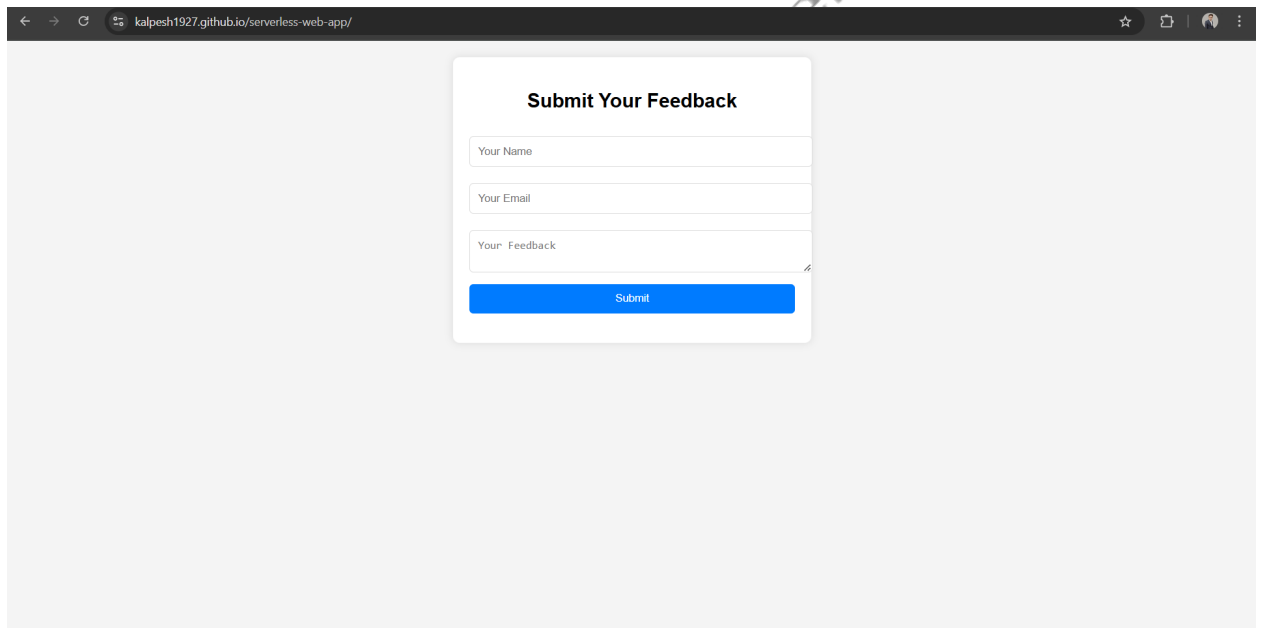


```
1 async function submitFeedback() {
2   const data = {
3     user_name: document.getElementById("name").value,
4     email: document.getElementById("email").value,
5     message: document.getElementById("message").value
6   };
7
8   const response = await fetch("https://691dtuvpl.execute-api.ap-south-1.amazonaws.com/prodfeedback/feedback", {
9     method: "POST",
10    headers: { "Content-Type": "application/json" },
11    body: JSON.stringify(data)
12  });
13
14  const result = await response.json();
15  document.getElementById("responseMessage").innerText = result.message;
16 }
```

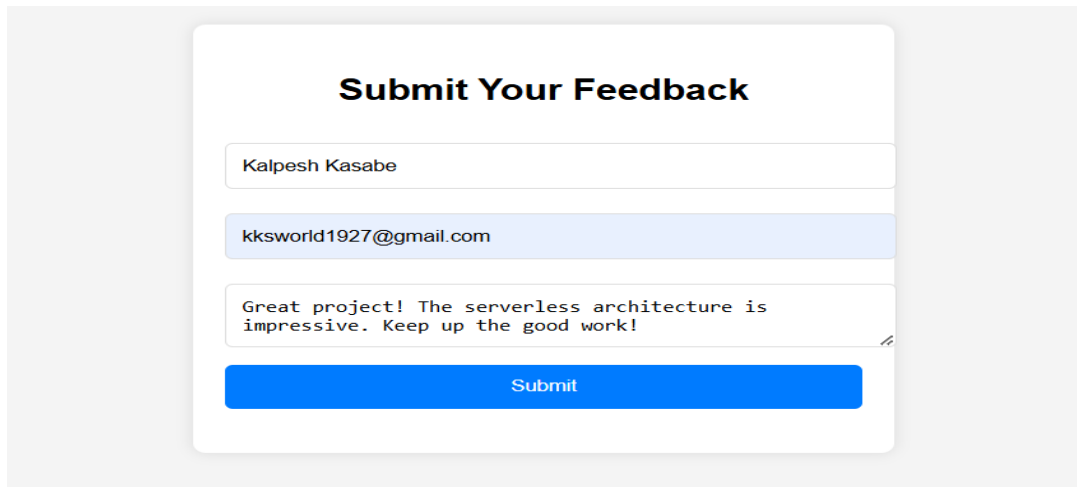
2. Upload the updated script.js to the GitHub repository.

## Step 6: Test and Deploy

1. Open the **GitHub Pages URL** .



2. Submit feedback via the form.



**Submit Your Feedback**

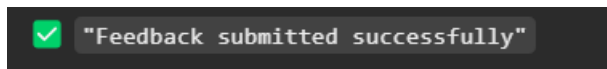
Kalpesh Kasabe

kksworld1927@gmail.com

Great project! The serverless architecture is impressive. Keep up the good work!

Submit

3. Check the **DynamoDB Table** for the new entry.



4. If successful, your serverless web app is live!

---

## Conclusion

You have successfully built and deployed a cost-free serverless web application using AWS and GitHub Pages. This project helped you understand GitHub static hosting, API Gateway, Lambda, and DynamoDB integration while staying within AWS Free Tier limits.

### How to Avoid AWS Costs Completely:

1. **Monitor AWS Usage** via the AWS Billing Dashboard.
2. **Delete the API Gateway & DynamoDB Table** if no longer needed.
3. **Use Local Development Tools** like AWS SAM or LocalStack for testing.

### Next Steps:

- Implement authentication using AWS Cognito (Free Tier Available).
- Add an email notification system using AWS SES (Free for first 1,000 emails/month).
- Enhance the frontend with better UI/UX.

[www.linkedin.com/in/kalpeshkasabe](http://www.linkedin.com/in/kalpeshkasabe)

[www.linkedin.com/in/kalpeshkasabe](http://www.linkedin.com/in/kalpeshkasabe)