

# **TUTORIAL ON HOW TO CREATE REACT APP AND INTEGRATE IT WITH AWS API GATEWAY**

INSTRUCTOR  
CLASS

## Introduction:

In this tutorial we are going to learn how to create a react web client and integrate it with aws api gateway. So we use aws amplify to deploy our react app and we create an aws lambda function using python programming language to integrate with api gateway.

### AWS Amplify

AWS Amplify is a set of purpose-built tools and features that lets frontend web and mobile developers quickly and easily build full-stack applications on AWS.

### AWS Lambda

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers

### AWS Api Gateway

Aws api gateway is a fully managed service which could be used to create restful apis.

## Prerequisites:

- Nodejs
- Python
- Aws CLI

If you do not have Nodejs already installed in your environment follow the link [here](#) and install the Nodejs. We use [npm](#) to create our react app because it is the easiest way to create the app without worrying about internal setups done using webpack and babel.

You need python3 installed in your local machine if it is already there you need to download it from [here](#) and install it.

Pip will be available once you have installed python. After that you will need to install aws cli if you have not already configured the aws cli in your local machine. In a terminal try to paste below and install aws cli.

**pip install awscli --upgrade --user**

After that you need to configure the aws then past below command and provide correct values which are related to your aws account

**aws configure**

You will get below to add the data. For your user which is used to login to aws console must have an access key and secret. You have to remember the secret value you have entered when you are creating the access key. Please login to aws console and search for IAM and open IAM console navigate to your user and you will find the place to create a key if you have not already done it. Please follow this [guide](#) to get more details.

AWS Access Key ID [None]: \*\*\*PASTE ACCESS KEY ID HERE\*\*\*

AWS Secret Access Key [None]: \*\*\*PASTE SECRET ACCESS KEY HERE\*\*\*

Default region name [None]: \*\*\*TYPE YOUR PREFERRED REGION\*\*\*

Default output format [None]: json

All good upto now right. Ok let's start creating our first restful api first.

## Creation of rest api:

As mentioned in the introduction we use aws lambda function to create backend functionality of our api. So login to aws console and search for aws lambda and navigate to aws lambda. Then click on create button to create a lambda function.

The screenshot shows the 'Create function' page in the AWS Lambda console. It features four tabs for creating a function: 'Author from scratch' (selected), 'Use a blueprint', 'Container image', and 'Browse serverless app repository'. Below the tabs is the 'Basic information' section, which includes a text input for 'Function name' (containing 'filmApiFunction'), a dropdown for 'Runtime' (set to 'Python 3.9'), and radio buttons for 'Architecture' (with 'x86\_64' selected and 'arm64' as an option).

Add a function name and runtime in my case I have selected python3.9. In permissions choose create new role with basic lambda permission.

**Permissions** [info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Create a new role with basic Lambda permissions  
☐ Use an existing role  
☐ Create a new role from AWS policy templates

ⓘ Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named `filmApiFunction-role-rvyw7hdy`, with permission to upload logs to Amazon CloudWatch Logs.

▼ **Advanced settings**

☐ **Enable Code signing** [info](#)  
 Use code signing configurations to ensure that the code has been signed by an approved source and has not been altered since signing.

☐ **Enable VPC** [info](#)  
 Connect your function to a VPC to access private resources during invocation.

Cancel **Create function**

After that click on the create function button. Ok now we have created the aws lambda function and we need to access the dynamodb from our lambda function for that we need to setup permission with the role we got created for the lambda. Let's navigate to the configuration tab and select permission part in left pane of lambda console and navigate to the role.

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration | Triggers | **Permissions** | Destinations | Environment variables | Tags | VPC | Monitoring and operations tools | Concurrency | Asynchronous invocation

**Execution role** [Edit](#)

Role name  
[filmApiFunction-role-rvyw7hdy](#)

**Resource summary** [View role document](#)

Amazon CloudWatch Logs  
3 actions, 2 resources

To view the resources and actions that your function has permission to access, choose a service.

By action | **By resource**

Resource	Actions
----------	---------

Once you click on the role name it will navigate to the IAM console for the role. There under the permission tab you have a button to Add permission. Click on it and select attach policies from the drop down.

**Permissions** | Trust relationships | Tags | Access Advisor | Revoke sessions

**Permissions policies (1)**  
You can attach up to 10 managed policies.

Filter policies by property or policy name and press enter

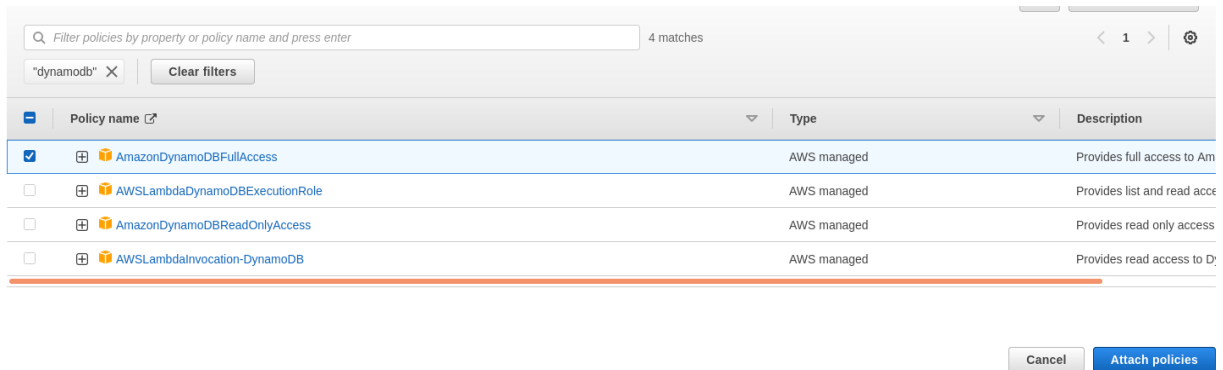
☐ Policy name [↗](#)

[Add permissions](#) ▲  
 Attach policies  
 Create inline policy

[Simulate](#) [Remove](#)

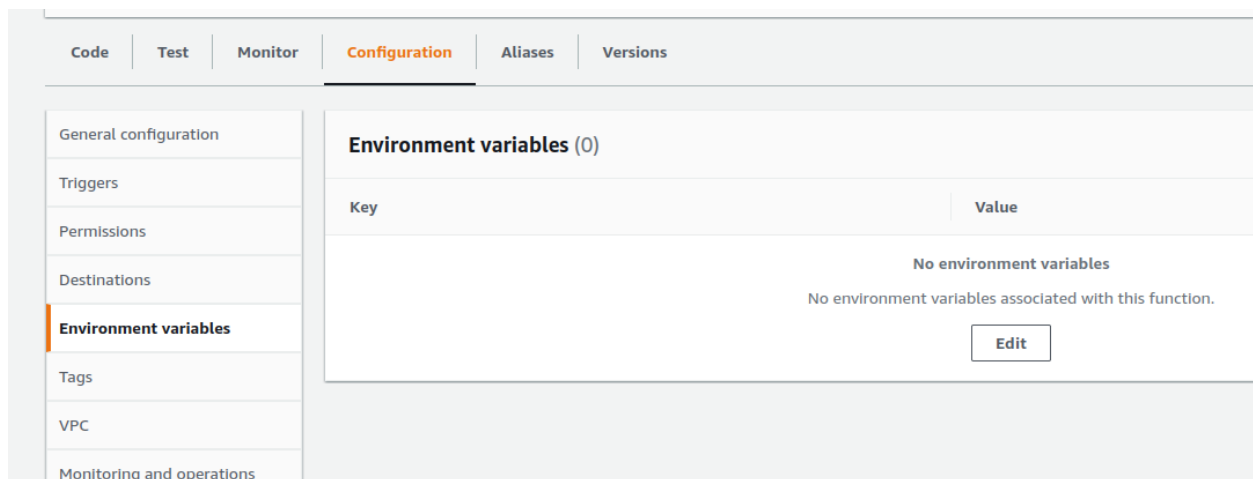
▼ Type

Then search for the dynamodb and select the full access permission set and click on attach policies.



Ok well done we have successfully attached the policy. Let's go back to lambda console and add an environment variable to the lambda. In here we are going to pass the dynamodb table name as an env variable.

Under configurations select the environment variable in the left pane.



After that click on the edit button to add a variable. And it will show below window there you should click on the Add environment variable button.

The screenshot shows the 'Edit environment variables' page in the AWS Lambda console. The breadcrumb trail is 'Lambda > Functions > filmApiFunction > Edit environment variables'. The page title is 'Edit environment variables'. Below the title is a section titled 'Environment variables' with a description: 'You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)'. Below this, it states 'There are no environment variables on this function.' and provides an 'Add environment variable' button. At the bottom of the main content area is an 'Encryption configuration' section with a right-pointing arrow. At the bottom right of the page are 'Cancel' and 'Save' buttons.

Add the variable named TABLENAME with your db name as the value and click on the save button.

This screenshot shows the 'Edit environment variables' page after adding a variable. The breadcrumb trail is 'Lambda > Functions > filmApiFunction > Edit environment variables'. The page title is 'Edit environment variables'. The 'Environment variables' section contains a table with one row:

Key	Value	
<input type="text" value="TABLENAME"/>	<input type="text" value="FilmInfoTable"/>	<input type="button" value="Remove"/>

Below the table is an 'Add environment variable' button. At the bottom of the main content area is an 'Encryption configuration' section with a right-pointing arrow. At the bottom right of the page are 'Cancel' and 'Save' buttons.

Ok we have added the table name but did not create it let's create our dynamoDB table.

Create a json file named db-table.json and add this content in it.

```
{
  "TableName": "FilmInfoTable",
  "KeySchema": [
    { "AttributeName": "Director", "KeyType": "HASH" },
    { "AttributeName": "FilmTitle", "KeyType": "RANGE" }
  ],
  "AttributeDefinitions": [
    { "AttributeName": "Director", "AttributeType": "S" },
    { "AttributeName": "FilmTitle", "AttributeType": "S" }
  ],
  "ProvisionedThroughput": {
    "ReadCapacityUnits": 5,
    "WriteCapacityUnits": 5
  }
}
```

Here we are creating a dynamodb table named FilmInfoTable and it's partition key is Director and the sort key is the FilmTitle

### 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.

### Sort key

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

More about the provisioned information can be found [here](#).

Let's create the table and add the data to the table now.

Run the below command in the terminal where your json file exists.

**aws dynamodb create-table --cli-input-json file://db-table.json**

You will see a output like this:

```
{
  "TableDescription": {
    "AttributeDefinitions": [
      {
        "AttributeName": "Director",
        "AttributeType": "S"
      },
      {
        "AttributeName": "FilmTitle",
        "AttributeType": "S"
      }
    ],
    "TableName": "FilmInfoTable",
    "KeySchema": [
      {
        "AttributeName": "Director",
        "KeyType": "HASH"
      },
      {
        "AttributeName": "FilmTitle",
        "KeyType": "RANGE"
      }
    ],
    "TableStatus": "CREATING",
    "CreationDateTime": 1648446951.308,
    "ProvisionedThroughput": {
      "NumberOfDecreasesToday": 0,
      "ReadCapacityUnits": 5,
      "WriteCapacityUnits": 5
    },
    "TableSizeBytes": 0,
    "ItemCount": 0,
    "TableArn": "arn:aws:dynamodb:eu-west-1:057186357501:table/FilmInfoTable",
    "TableId": "edcc9375-36f5-49ac-bd4c-7dde8e425649"
  }
}
```

You can navigate to aws console and search for the dynamodb and go dynamodb and see the table which you created.

The screenshot shows the AWS DynamoDB console interface. On the left is a sidebar with navigation links: Dashboard, Tables (selected), Update settings, Explore items, PartiQL editor, Backups, Exports to S3, and Reserved capacity. The main area is titled 'DynamoDB > Tables' and shows a list of tables. At the top right of the table list are buttons for 'Create table', 'Delete', and 'Actions'. Below the buttons is a search bar and a dropdown for 'Any table tag'. The table list has columns: Name, Status, Partition key, Sort key, Indexes, Read capacity mode, Write capacity mode, Size, and Table class. One table is listed: 'FilmInfoTable' with status 'Active', partition key 'Director (S)', sort key 'FilmTitle (S)', 0 indexes, 'Provisioned (S)' for both read and write capacity, 0 bytes size, and 'DynamoDB Standard' table class.

	Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Size	Table class
<input type="checkbox"/>	FilmInfoTable	Active	Director (S)	FilmTitle (S)	0	Provisioned (S)	Provisioned (S)	0 bytes	DynamoDB Standard



Let's add some data to this table now. Create a json file named test-data.json and add the below content in it.

```
{

  "FilmInfoTable": [{

    "PutRequest": {

      "Item": {

        "Director": {

          "S": "ADAM"

        },

        "FilmTitle": {

          "S": "Sleep less"

        }

      }

    }

  },

  {

    "PutRequest": {

      "Item": {

        "Director": {

          "S": "Stefan"

        },

        "FilmTitle": {

          "S": "Paint ball"

        }

      }

    }

  }

}
```

```

    }

    }

  },

  {

    "PutRequest": {

      "Item": {

        "Director": {

          "S": "Lisa"

        },

        "FilmTitle": {

          "S": "Fedup"

        }

      }

    }

  }

]
}

```

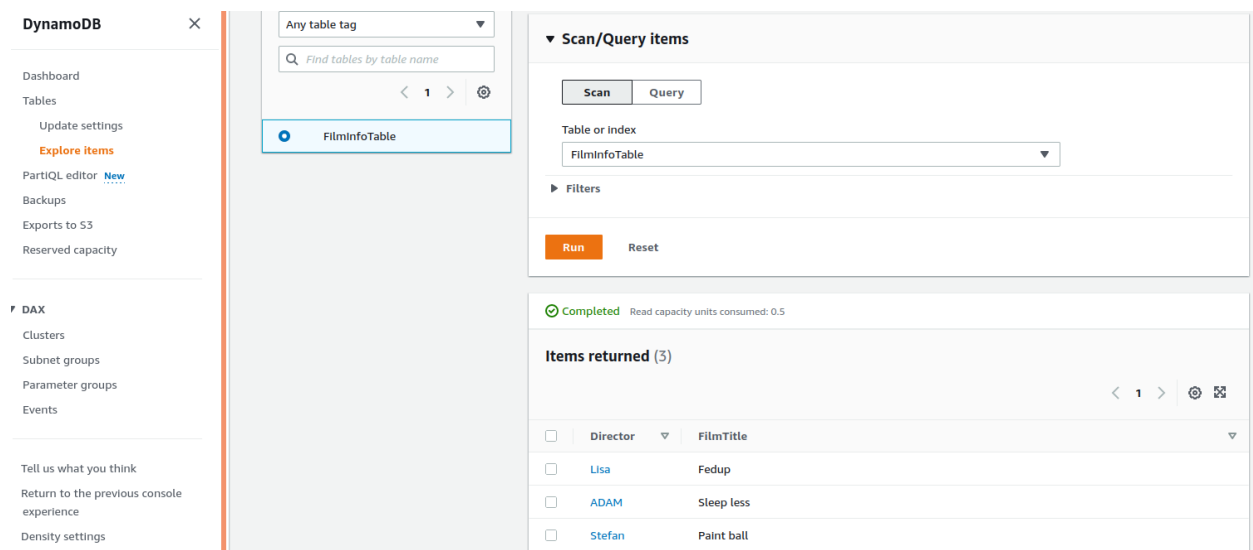
Now execute below command.

**aws dynamodb batch-write-item --request-items file:///test-data.json**

You should see a result like this

```
{"UnprocessedItems": {}}
```

If you explore the items in the newly created table you will see the inserted data.



That's all about the configuration. Let's start coding our function. Navigate back to the code tab and add the below code to your function.

```
import json
import boto3
import os

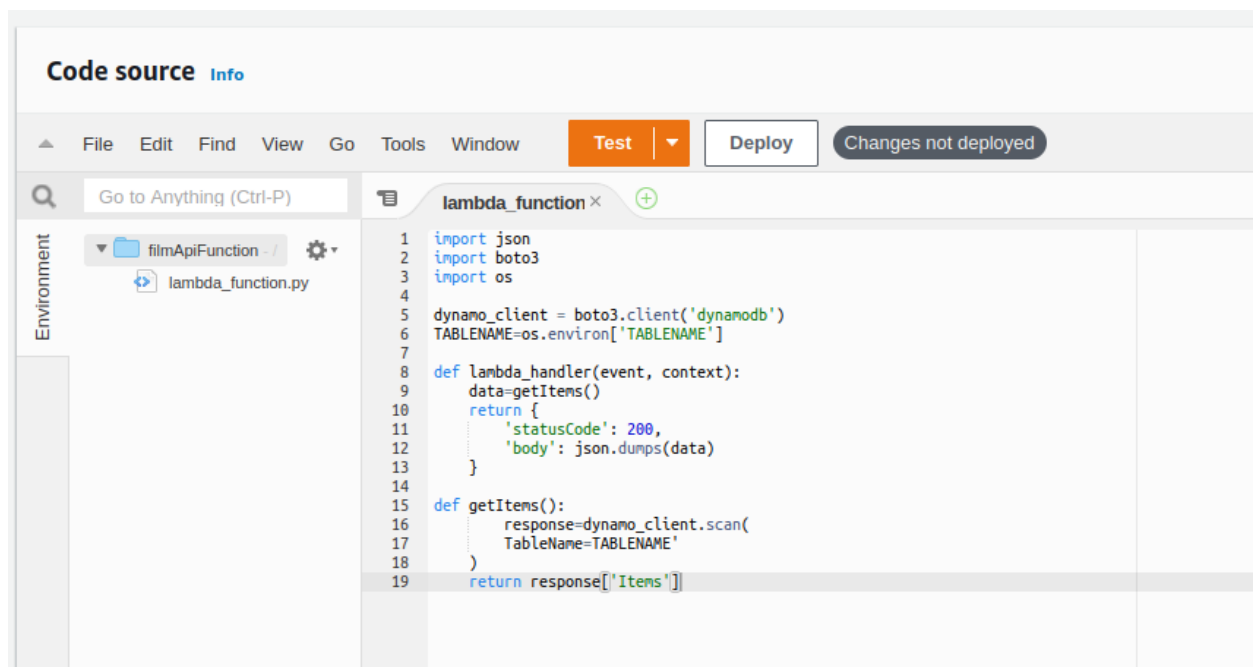
dynamo_client = boto3.client('dynamodb')
TABLENAME=os.environ['TABLENAME']

def lambda_handler(event, context):
    data=getItems()
    return {
```

```
'statusCode': 200,  
'body': json.dumps(data)  
}
```

```
def getItems():  
    response=dynamo_client.scan(  
        TableName=TABLENAME  
    )  
    return response['Items']
```

Now you have to click on the deploy button.



Once you deploy your code we can test our code click on test button and add below configurations as in the image and click on save.

The screenshot shows the 'Test' configuration page for a Lambda function. At the top, there are two buttons: 'Create new event' (active) and 'Edit saved event'. Below this is the 'Event name' field, which contains the text 'test'. A note below the field states: 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.' Under 'Event sharing settings', the 'Private' radio button is selected, with a note: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)'. The 'Shareable' option is also present with a note: 'This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)'. The 'Template - optional' dropdown menu is set to 'hello-world'. At the bottom, there is a section for 'Event JSON' with a 'Format JSON' button. The JSON content is partially visible as '{' on line 1 and '}' on line 2.

After that again click on the test button. So you will see the result of the function execution as below.

The screenshot shows the 'Execution results' tab for the Lambda function. It indicates the status is 'Succeeded', with 'Max memory used: 67 MB' and 'Time: 220.62 ms'. A message states: 'A function update is still in progress so the invocation went to the previously deployed code and configuration.' The 'Test Event Name' is 'test'. The 'Response' is a JSON object: 

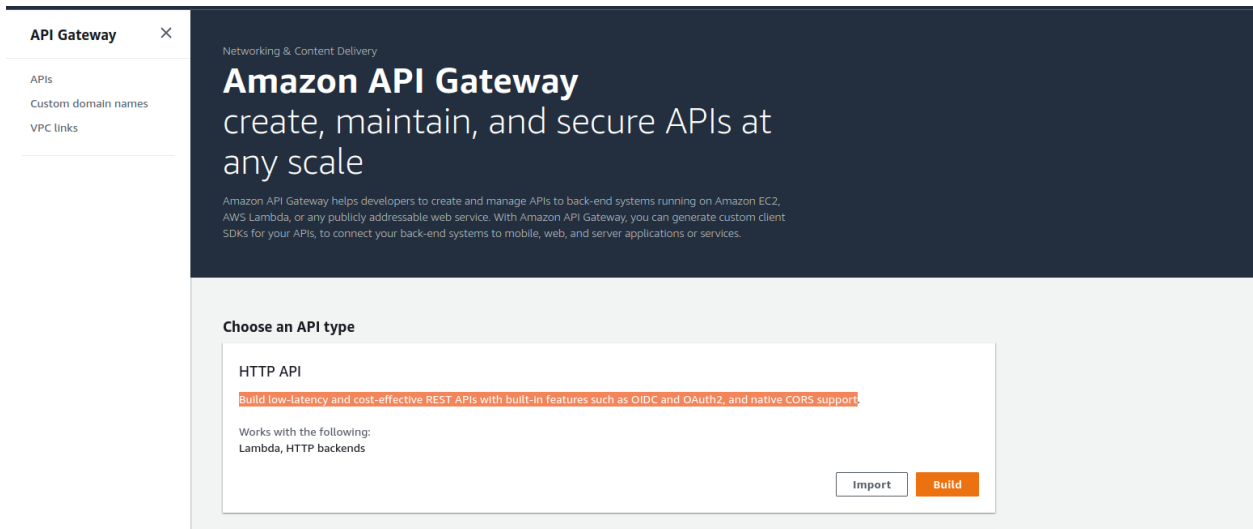
```
{  "statusCode": 200,  "body": "[{"Director": {"S": "Lisa"}, "FilmTitle": {"S": "Fedup"}}, {"Director": {"S": "Adam"}, "FilmTitle": {"S": "Sleep less"}}, {"Director": {"S": "Stefan"}, "FilmTitle": {"S": "The Grand Budapest Hotel"}]"}]
```

. The 'Function Logs' section shows the following details: 

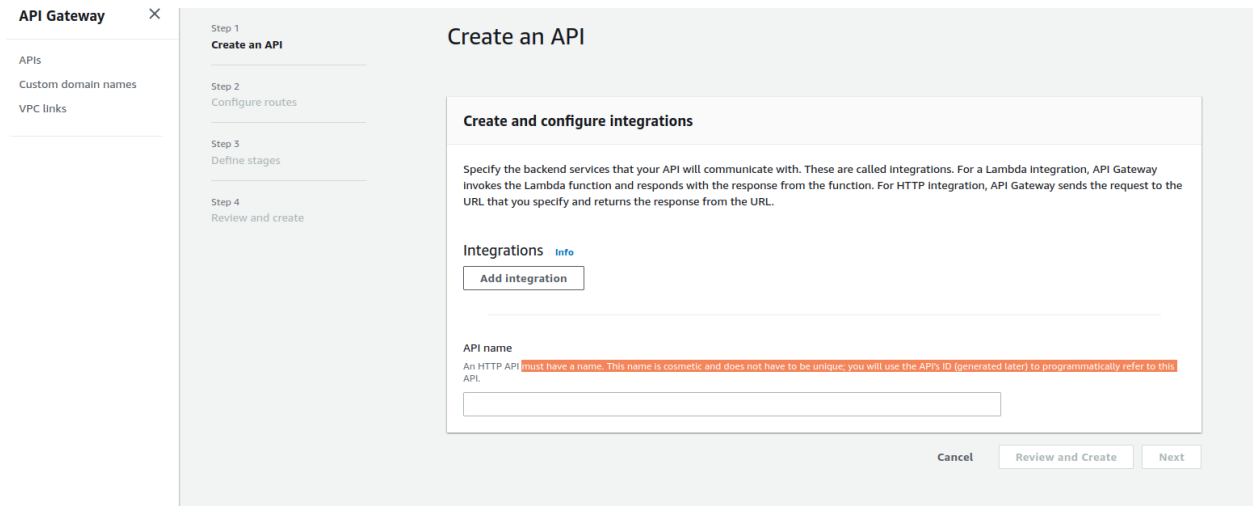
```
START RequestId: 81a69488-d992-448d-9957-1beac99c6282 Version: $LATEST
END RequestId: 81a69488-d992-448d-9957-1beac99c6282
REPORT RequestId: 81a69488-d992-448d-9957-1beac99c6282  Duration: 220.62 ms Billed Duration: 221 ms Memory Size: 128 MB Max Memory Used: 67 MB Init Duration: 326.16 ms
```

. The 'Request ID' is '81a69488-d992-448d-9957-1beac99c6282'.

Ok we can see the results were returned correctly. Then let's create the api gateway to integrate with lambda. Search for the apigateway and navigate to api gateway in aws console. And click on create api gateway. There you can use HTTP Api and click on build button.



Once you click on build you will see the window below.



Click on add integration then and add the details as below. Here you select the lambda function you have already created in previous step.

**Create an API**

Step 1: Create an API  
Step 2: **Configure routes**  
Step 3: Define stages  
Step 4: Review and create

### Create and configure integrations

Specify the backend services that your API will communicate with. These are called Integrations. For a Lambda integration, API Gateway invokes the Lambda function and responds with the response from the function. For HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

**Integrations** [Info](#)

Lambda Remove

AWS Region: eu-west-1  
Lambda function: filmApiFunction  
Version: 2.0 [Learn more.](#)

Add integration

**API name**  
An HTTP API must have a name. This name is cosmetic and does not have to be unique; you will use the API's ID (generated later) to programmatically refer to this API.

filmapi

Cancel Review and Create Next

Then click next.

**API Gateway** ×

APIs  
Custom domain names  
VPC links

Step 1: Create an API  
Step 2: Configure routes  
Step 3: **Define stages**  
Step 4: Review and create

### Configure routes

**Configure routes** [Info](#)

API Gateway uses routes to expose integrations to consumers of your API. Routes for HTTP APIs consist of two parts: an HTTP method and a resource path (e.g., GET /pets). You can define specific HTTP methods for your integration (GET, POST, PUT, PATCH, HEAD, OPTIONS, and DELETE) or use the ANY method to match all methods that you haven't defined on a given resource.

Method: GET  
Resource path: /filmApiFunction  
Integration target: filmApiFunction Remove

Add route

Cancel Previous Next

Add the route method as Get and click next.

Step 1

Create an API

Step 2

Configure routes

Step 3

**Define stages**

Step 4

Review and create

## Define stages

### Configure stages [Info](#)

Stages are independently configurable environments that your API can be deployed to. You must deploy to a stage for API configuration changes to take effect, unless that stage is configured to autodeploy. By default, all HTTP APIs created through the console have a default stage named \$default. All changes that you make to your API are autodeployed to that stage. You can add stages that represent environments such as development or production.

Stage name

Auto-deploy ☒

Remove

Add stage

Cancel Previous Next

Keep the default on stage and click next.

Step 4

**Review and create**

API name

filmapi

Integrations

filmApiFunction (Lambda)

Routes

Edit

Routes

GET /filmApiFunction → filmApiFunction (Lambda)

Stages

Edit

Stages

\$default (Auto-deploy: enabled)

Cancel Previous Create



Now click on create to create api gateway.

**filmapi** Edit

**API details**

API ID	Protocol	Created
0fhnl08j5d	HTTP	2022-03-30
Description	Default endpoint	
No Description	Enabled	

**Stages for filmapi**

Stage name	Invoke URL	Attached deployment	Auto deploy	Last updated
\$default	<a href="https://0fhnl08j5d.execute-api.eu-west-1.amazonaws.com">https://0fhnl08j5d.execute-api.eu-west-1.amazonaws.com</a>	da8mo6	enabled	2022-03-30

Here you will see the invoke url . You can now test the api with the route you created. So the link will look like this.

<invoke url>/filmApiFunction

[←](#) [→](#) [↺](#) [🔒](#) <https://0fhnl08j5d.execute-api.eu-west-1.amazonaws.com/filmApiFunction> [☆](#) [🔔](#)

Getting Started Mini Cooper Morris 19... Learn GIMP Tutorial - ... FooBar Serverless - Y... 10 Best Casino Affiliat... Software Affiliate Pro... <https://www.blackma...>

[{"Director": {"S": "Lisa"}, "FilmTitle": {"S": "Fedup"}}, {"Director": {"S": "ADAM"}, "FilmTitle": {"S": "Sleep less"}}, {"Director": {"S": "Stefan"}, "FilmTitle": {"S": "Paint ball"}}]

Now our api is up and running we can see the results. But still we need to enable cors to access the api from react app so in apigateway navigate to cors part and add below configuration.

**API Gateway** ×

**APIs**  
Custom domain names  
VPC links

API: filmapi (0fhnl08j5d)

**Develop**  
Routes  
Authorization  
Integrations  
**CORS**  
Reimport  
Export

**Deploy**  
Stages

**Protect**  
Throttling

**API Gateway** > CORS Stage: - ▼ Deploy

**Cross-Origin Resource Sharing**

**Configure CORS** Info Configure Clear

CORS allows resources from different domains to be loaded by browsers. If you configure CORS for an API, API Gateway ignores CORS headers returned from your backend integration. See our [CORS documentation](#) for more details.

Access-Control-Allow-Origin	Access-Control-Allow-Headers
<input type="text" value="*"/>	No Headers are allowed
Access-Control-Allow-Methods	Access-Control-Expose-Headers
<input type="text" value="GET"/>	No Expose Headers are allowed
Access-Control-Max-Age	Access-Control-Allow-Credentials
0 Seconds	<input type="checkbox"/> NO

Let's create our react app as the next step.

## Creation of react app:

Open a terminal in the machine in a preferred location and type below command to create a react app.

**npx create-react-app film-site**

You will see a result like this at the end.

```
Created git commit.
Success! Created film-site at /home/dushan/Documents/tutorial2/film-site
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

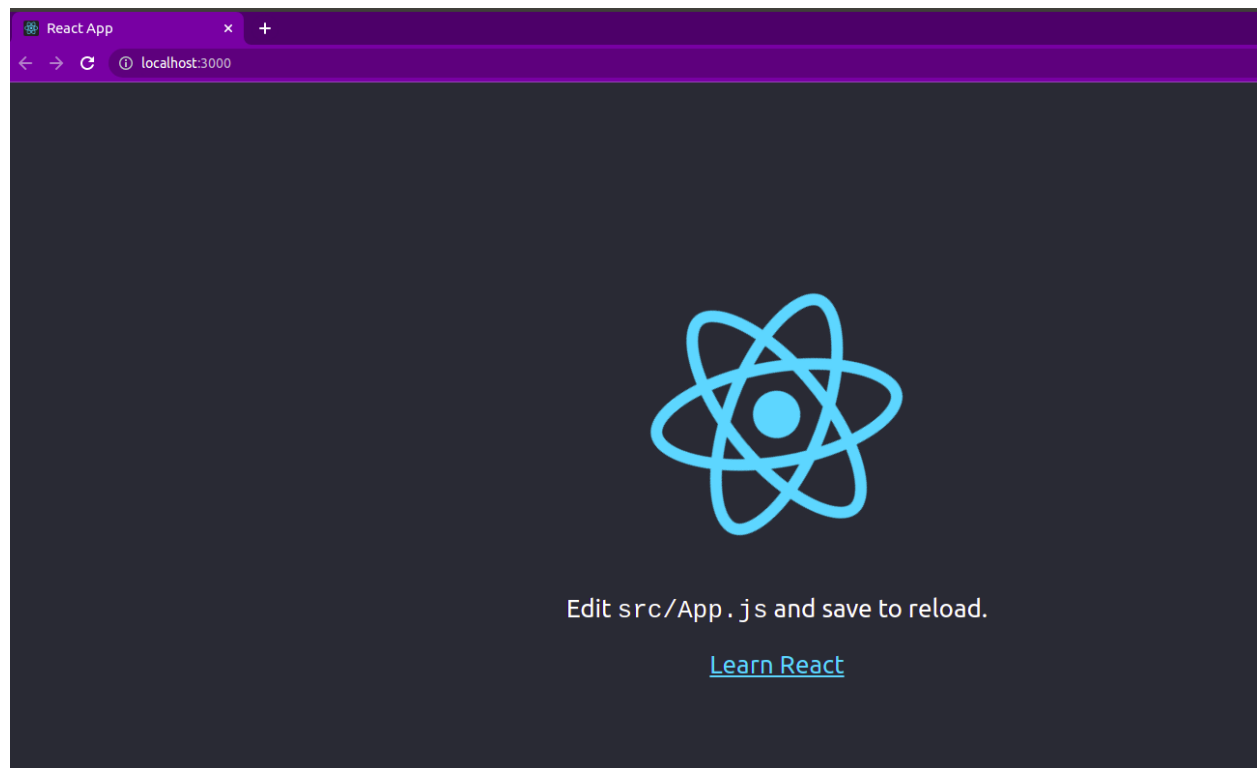
  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

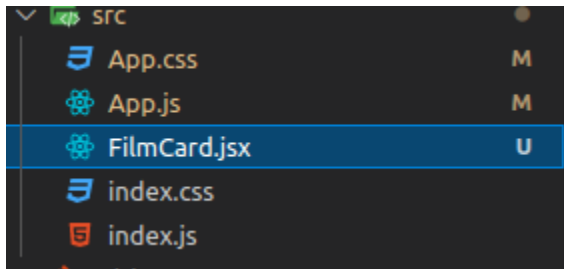
  cd film-site
  npm start

Happy hacking!
npm notice
npm notice New patch version of npm available! 8.5.1 -> 8.5.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.5.5
npm notice Run npm install -g npm@8.5.5 to update!
npm notice
```

Now navigate to file-site in terminal by **cd film-site** and in there run **npm start** and see the result.If you see below result the app created working properly in your local environment.



Then let's clean the src folder a bit and start creating files. I have created a new file named FilmCard.jsx. And removed several files from example see below folder structure.



Now Let's edit our App.js file to visualize our data. We use the useEffect hook to fetch the data from our api gateway when the web is initialized. Then we display the information on a card.

```
import React from 'react';
import './App.css';
import { useState, useEffect } from 'react'
import FilmCard from './FilmCard';

const API_URL =
'https://0fhnl08j5d.execute-api.eu-west-1.amazonaws.com/filmApiFunction';

function App() {
  const movie1={"Director": {"S": "Lisa"}, "FilmTitle": {"S": "Fedup"}}
  const [films, setFilms] = useState([])
  const movieList = async() => {
    const response = await fetch(API_URL);
    const data = await response.json();
    setFilms(data)
    console.log(data);
  };
  useEffect(() => {
    movieList();
  }, []);

  return (
    <div className = "App">
      <h1> Movie Flix </h1>
      {
        films?.length > 0
        ? (
```

```

        <div className='container'>
          {films.map((film) => (
            <FilmCard movie={film}/>
          ))}
        </div>
      ):
      (
        <div>
          <p>Movies not found</p>
        </div>
      )
    }
  </div>
);
}

export default App;

```

I have created a separate component to handle data of the card open newly created FilmCard.jsx file and add below content.

```

import React from "react"

const FilmCard = ({movie}) => {
  return (
    <div className='movie'>
      <p>{movie['Director']['S']}</p>
      <div>
        <img src='https://via.placeholder.com/400'></img>
      </div>
      <div>
        <h3>{movie['FilmTitle']['S']}</h3>
      </div>
    </div>
  );
}

export default FilmCard;

```

Then let's add some css to our project using app.css.

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

.movie {
  width: 310px;
  height: 460px;
  margin: 1.5rem;
  position: relative;
  border-radius: 12px;
  overflow: hidden;
  border: none;
  transition: all 0.4s cubic-bezier(0.175, 0.885, 0, 1);
  box-shadow: 0px 13px 10px -7px rgba(0, 0, 0, 0.1);
}
```

```
}

.movie div:nth-of-type(1) {
  width: 100%;
  height: 100%;
}

.movie div:nth-of-type(1) img {
  height: 100%;
  width: 100%;
}

.movie div:nth-of-type(2) {
  z-index: 2;
  background-color: #343739;
  padding: 16px 24px 24px 24px;
  position: absolute;
  bottom: 0;
  right: 0;
  left: 0;
  font-family: "Roboto Slab", serif;
  color: #f9d3b4;
}

.container {
  width: 100%;
  margin-top: 3rem;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-wrap: wrap;
}

.movie div:nth-of-type(3) {
  z-index: 2;
  background-color: #343739;
  padding: 16px 24px 24px 24px;
  position: absolute;
  bottom: 0;
  right: 0;
```

```

    left: 0;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

Ok now we need to change the header of our app. Navigate to public folder and edit the index.html file> head> title to 'Movie Flix'

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>Movie Flix</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>

  </body>
</html>

```

Our Index.js file looks like this.

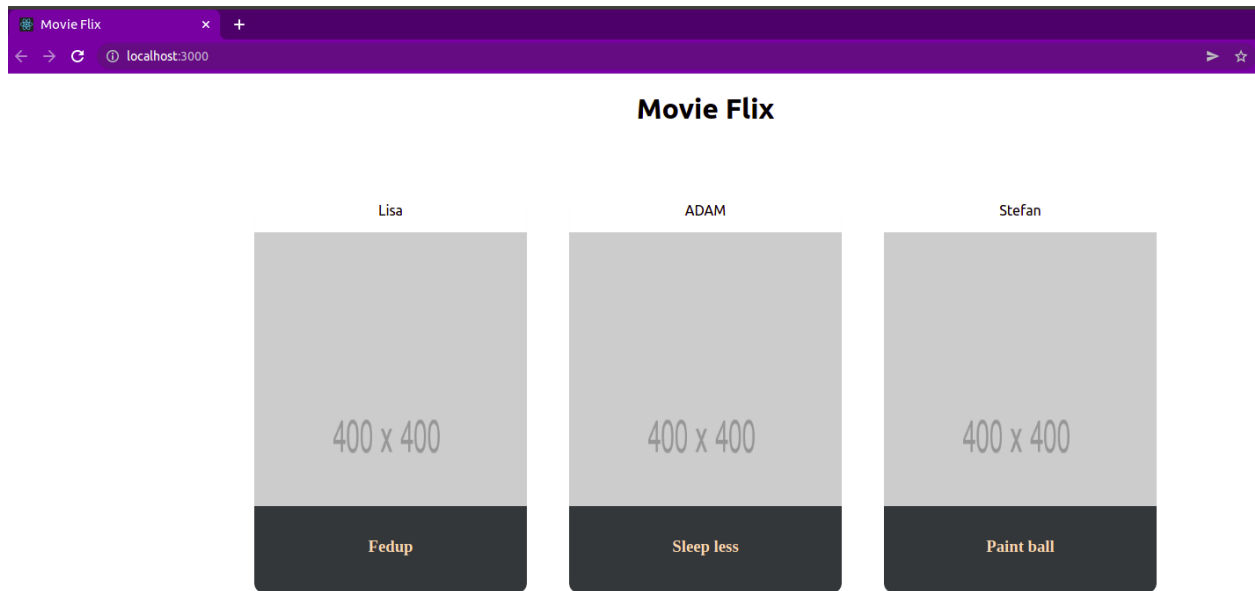
```

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';

```

```
import App from './App';
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>, document.getElementById('root') );
```

Time to see our changes in the local environment. In terminal type `npm start`



Ok now let's build our project using below command  
**npm run build**

You will see a result like this and a build folder will get created in the root directory.

```
> film-site@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled with warnings.

src/App.js
  Line 9:11: 'moviel' is assigned a value but never used  no-unused-vars
src/FilmCard.jsx
  Line 8:9:  img elements must have an alt prop, either with meaningful text, or an empty string for decorative images  jsx-ally/alt-text

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

File sizes after gzip:

 48.15 kB  build/static/js/main.a9adb5dd.js
 810 B     build/static/css/main.bade1ecb.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

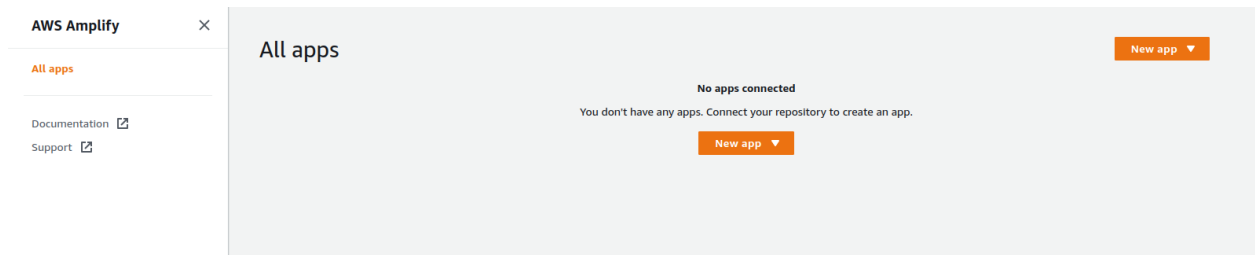
  https://cra.link/deployment

npm notice
npm notice New patch version of npm available! 8.5.1 -> 8.5.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.5.5
```

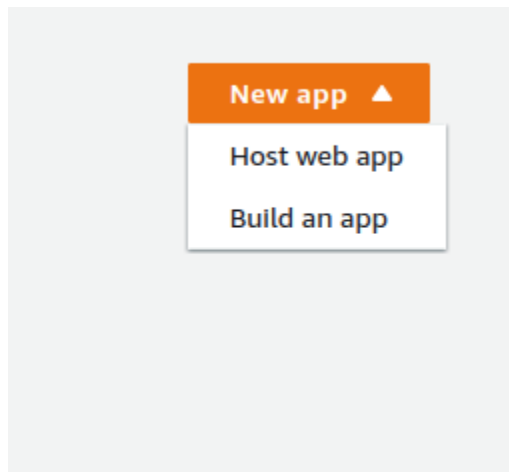


## Deploy React app to aws amplify:

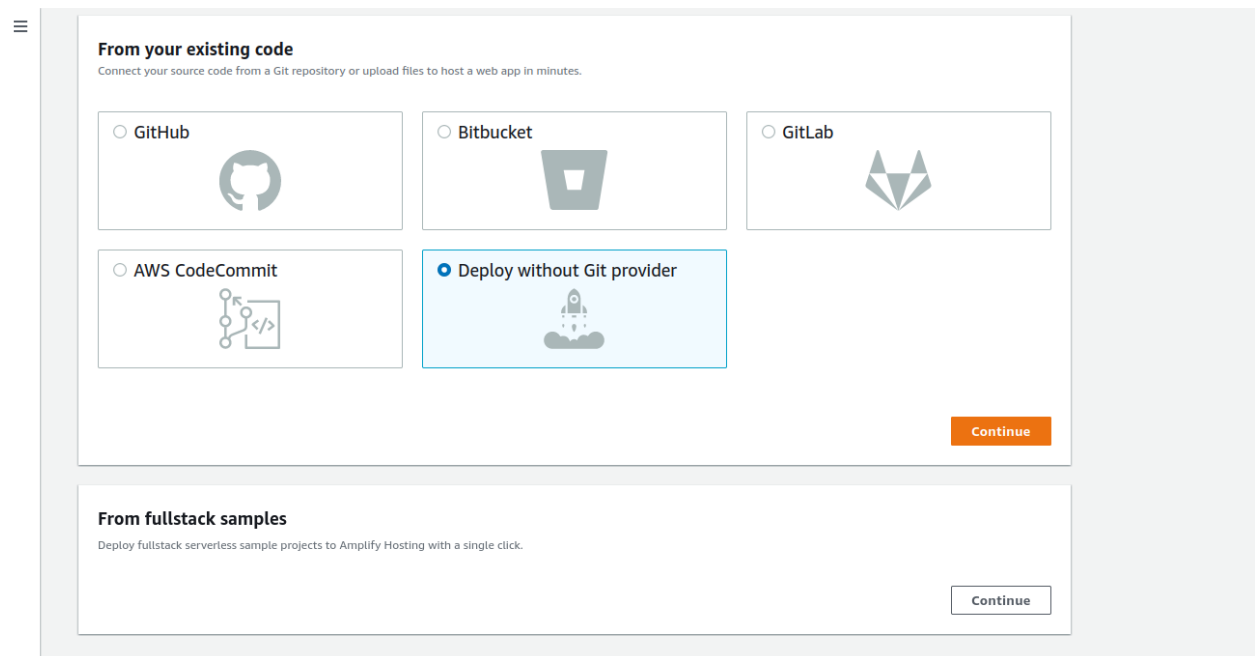
In aws console navigate to aws amplify.



Then click on create new app button and select Host web app from drop down.



In here I'm planning to upload the build folder straightforwardly to amplify so I'm selecting Deploy without git provider.



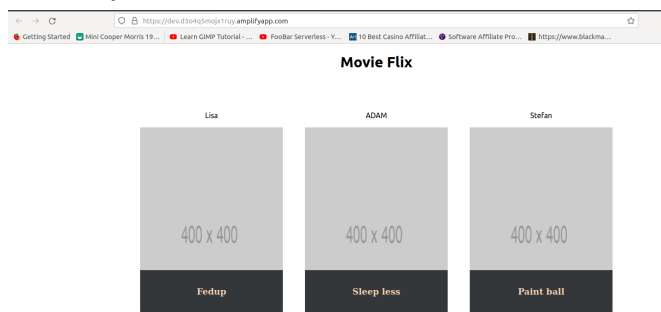
In the next window add an app name and an environment name as you wish and drag and drop the build folder of your react project.

The screenshot shows the AWS Amplify console interface. On the left is a sidebar with 'AWS Amplify' at the top, followed by 'All apps', 'Documentation', and 'Support'. The main area is titled 'Manually upload objects to deploy your app. You can choose to drag and drop the artifacts directly, pull a zip from an existing S3 bucket or any other URL.' Below this is a 'Start a manual deployment' form. The form has two input fields: 'App name' with the value 'filmsite' and 'Environment name' with the value 'dev'. Under the 'Method' section, there are three options: 'Drag and drop' (selected with a blue circle), 'Amazon S3', and 'Any URL'. Below these is a large light blue box with a file icon and the text '/build'. At the bottom right of the form are three buttons: 'Cancel', 'Previous', and 'Save and deploy'.

Once you upload click on save and deploy button.

The screenshot shows the AWS Amplify console after a successful deployment. The sidebar is the same as in the previous image. The main area has a header 'This tab lists all connected branches, select a branch to view build details.' and an 'Add environment' button. Below this is a section for the 'dev' environment. It features a placeholder image of a browser window with the Amazon logo. To the right, it says 'Deployment successfully completed.' with a green progress bar at 100%. Below the progress bar, it shows the 'Domain' as 'https://dev.d3o4q5mojx1ruy.amplifyapp.com' and the 'Last deployment' as '3/30/2022, 11:28:47 AM'. At the bottom, there is a large box with a hand icon and the text 'Drag and drop your project's build output directory or zip file here to update your app, or, choose another method.' and a 'Choose files' button.

Once the deployment is successful. You can navigate to your web page using the domain provided.



Ok We are done with our React sample app deployment in aws amplify.

**THE END.**