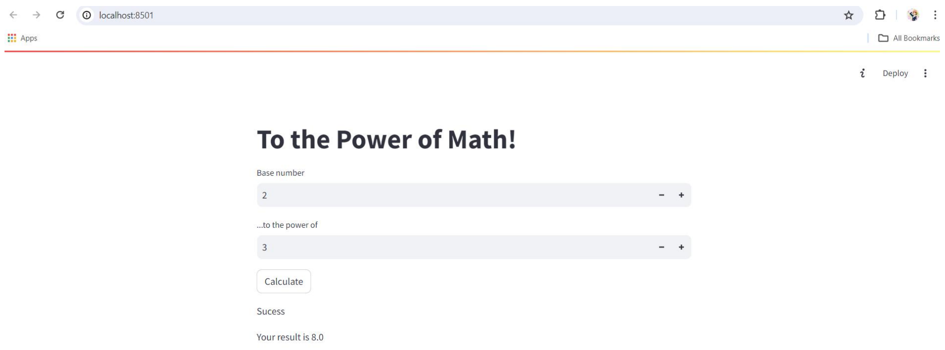
Creating a server-less web application on AWS :

Method 1 -

Using command line interface



To the Power of Math!

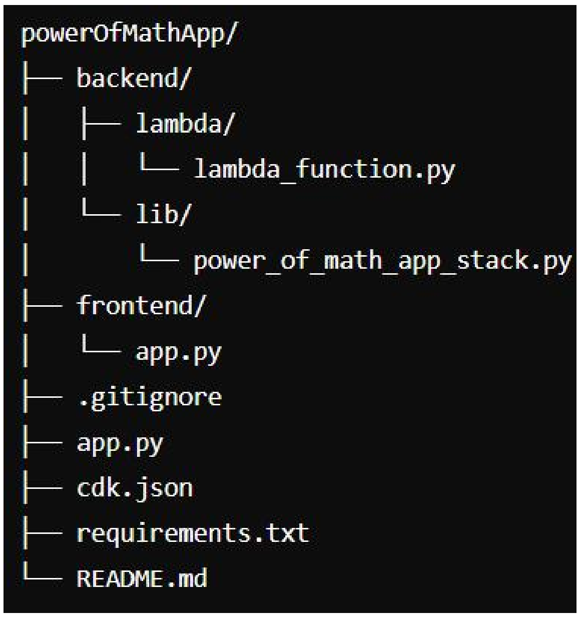
Base number: 2

...to the power of: 3

Sucess

Your result is 8.0

This project demonstrates a simple application that calculates the power of a number using a frontend built with Streamlit and a backend hosted on AWS using Lambda, DynamoDB, and API Gateway.



```
powerOfMathApp/
├── backend/
│   ├── lambda/
│   │   └── lambda_function.py
│   └── lib/
│       └── power_of_math_app_stack.py
├── frontend/
│   └── app.py
├── .gitignore
├── app.py
├── cdk.json
├── requirements.txt
└── README.md
```

Backend setup -

1. Initialize a Virtual Environment

First, we need to create a virtual environment to manage our dependencies. Run the following command in your project directory:

python -m venv .env

This command creates a directory named .env which contains a standalone Python installation and all necessary packages.

2. Activate the Virtual environment

Activate the virtual environment. On Windows, use:

```
.env\Scripts\activate
```

Activating the virtual environment ensures that all Python packages installed are local to this project and do not affect other projects on your system.

3. Install Required Packages

Install the necessary packages using pip:

```
pip install aws-cdk-lib constructs streamlit requests
```

- aws-cdk-lib: AWS Cloud Development Kit (CDK) library.
- constructs: Constructs are a basic building block for AWS CDK apps.
- streamlit: A library to create web apps easily.
- requests: A simple HTTP library for Python.

Create a requirements.txt file with the following content to ensure all dependencies are tracked:

```
aws-cdk-lib

constructs

streamlit

requests
```

4. Initialize CDK Project

Initialize the AWS CDK project. Navigate to the backend directory and run:

```
mkdir backend
cd backend
cdk init app --language python
```

This command initializes a new CDK application in the specified language (Python in this case).

5. Define the CDK Stack

Create a directory named lib inside backend and add a file named power_of_math_app_stack.py. This file will define your AWS infrastructure:

```python
from aws_cdk import (
    Stack,
    aws_dynamodb as dynamodb,
    aws_lambda as _lambda,
    aws_apigateway as apigateway,
    CfnOutput,
    RemovalPolicy
)
from constructs import Construct

class PowerOfMathAppStack(Stack):
    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # Create DynamoDB table
        table = dynamodb.Table(self, "PowerOfMathTable",
            partition_key=dynamodb.Attribute(name="ID", type=dynamodb.AttributeType.STRING),
            removal_policy=RemovalPolicy.DESTROY
        )

        # Create Lambda function
        lambda_function = _lambda.Function(self, "PowerOfMathFunction",
            runtime=_lambda.Runtime.PYTHON_3_8,
            handler="lambda_function.lambda_handler",
            code=_lambda.Code.from_asset("lambda"),
            environment={
                'TABLE_NAME': table.table_name
            }
        )

        # Grant the Lambda function read/write permissions to the DynamoDB table
        table.grant_read_write_data(lambda_function)

        # Create API Gateway
        api = apigateway.RestApi(self, "PowerOfMathAPI",
            rest_api_name="Power Of Math Service",
            description="This service calculates the power of a number."
        )

        # Create a resource and method for the API
        calculate_resource = api.root.add_resource("calculate")
        calculate_resource.add_method("POST",
            apigateway.LambdaIntegration(lambda_function),
            authorization_type=apigateway.AuthorizationType.NONE
        )

        # Output the API endpoint URL
        api_url_output = CfnOutput(self, "APIUrl",
            value=api.url,
            description="The URL of the API Gateway endpoint"
        )
```

This code defines a CDK stack that sets up:

- A DynamoDB table named PowerOfMathTable.
- A Lambda function named PowerOfMathFunction that handles the backend logic.
- An API Gateway to expose the Lambda function as a REST API.

6. Create Lambda Function

Create the lambda directory inside backend and add a file named lambda_function.py with the following content:

```
import json
import math
import boto3
import os
from time import gmtime, strftime
```

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table(os.environ['TABLE_NAME'])
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

def lambda_handler(event, context):
    body = json.loads(event['body'])
    base = int(body['base'])
    exponent = int(body['exponent'])
    math_result = math.pow(base, exponent)
    response = table.put_item(
        Item={
            'ID': str(math_result),
            'LatestGreetingTime': now
        })
    return {
        'statusCode': 200,
        'body': json.dumps(f'Your result is {math_result}')
    }
```

This code defines a Lambda function that reads a base and an exponent from the request, calculates the power, stores the result in DynamoDB, and returns the result.

7. Define the CDK App

Ensure your app.py in the backend directory looks like this:

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

import aws_cdk as cdk

from power_of_math_app.power_of_math_app_stack import PowerOfMathAppStack


app = cdk.App()
PowerOfMathAppStack(app, "PowerOfMathAppStack",

    )

app.synth()
```

This code initializes and synthesizes the CDK app, deploying the stack defined in power_of_math_app_stack.py.

8. Deploy the CDK Stack

Bootstrap CDK: *cdk bootstrap*

This command sets up the necessary resources for deploying CDK apps.

Deploy the Stack: *cdk deploy*

This command deploys the stack to your AWS account.

Frontend Setup

1. Create Streamlit App

Create a new directory named frontend and within it, create an app.py file.

2. Streamlit App Code

Add the following code to frontend/app.py:

```python
import streamlit as st
import requests

st.title('To the Power of Math!')

base = st.number_input('Base number', min_value=0, value=2)
exponent = st.number_input('...to the power of', min_value=0, value=3)

if st.button('Calculate'):
    api_gateway_url = "https://c2ocj3j0fe.execute-api.us-east-2.amazonaws.com/prod/calculate"
    response = requests.post(api_gateway_url, json={"base": base, "exponent": exponent})

    if response.status_code == 200:
        try:
            st.write("Success")
            response_json = response.json()
            st.write(response_json)
        except ValueError:
            st.error('Failed to parse JSON response')
    else:
        st.error(f'Error: {response.status_code}')
```

3. Running Your Streamlit App:

Navigate to the frontend directory: *cd frontend*

Run the Streamlit app: *streamlit run app.py*

This command starts the Streamlit app and provides a local URL (e.g., http://localhost:8501) to interact with the frontend.

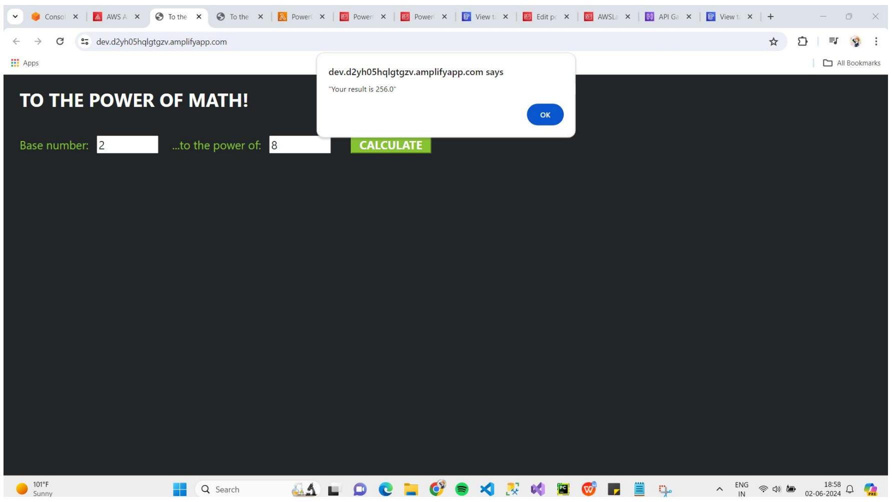
Method 2 -

Through AWS Console

Power of Math

Abstract -

This document outlines the process and logic for creating a server-less application on AWS called "Power of Math." Utilizing AWS services such as AWS Amplify, AWS Lambda, Amazon API Gateway, Amazon DynamoDB, and AWS Identity and Access Management (IAM). The project will involve creating and hosting a web page, performing mathematical operations, storing and retrieving results, and managing permissions securely.


TO THE POWER OF MATH!

Base number: 2 ...to the power of: 8 CALCULATE

dev.d2yh05hqlgtgzv.amplifyapp.com says

"Your result is 256.0"

OK

AWS Services Overview -

AWS Amplify: Used to build and host the front-end web application.

AWS Lambda: Runs backend code for mathematical operations.

Amazon API Gateway: Creates and manages API endpoints to invoke Lambda functions.

Amazon DynamoDB: Stores and retrieves math results.

AWS IAM: Manages permissions for secure access to AWS resources.

Project Requirements

AWS Account and access to the AWS Management Console.

A text editor (e.g., Notepad++).

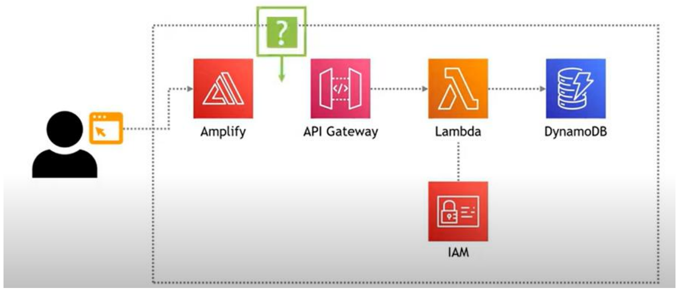
Hosting a webpage.

Invoking math functionality.

Storing and returning math results.

Handling permissions.

Architecture -



Work Flow -

1. Hosting a webpage -

Use AWS Amplify to host the webpage.

Create a simple HTML file (index.html) for the frontend.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>To the Power of Math!</title>
</head>
<body>
    To the Power of Math!
</body>
</html>
```

Deploy the web application using Amplify with the project name PowerOfMath and environment name dev.

2. **Creating the Lambda Function:**

Create a Lambda function named PowerOfMath to perform mathematical operations.

Lambda function code :

```
import json
import math

def lambda_handler(event, context):
    mathResult = math.pow(int(event['base']), int(event['exponent']))
    return {
        'statusCode': 200,
        'body': json.dumps('Your result is ' + str(mathResult))
    }
```

3. Setting Up API Gateway:

Create a REST API using Amazon API Gateway.

Create a POST method and integrate it with the PowerOfMath Lambda function.

Enable CORS to allow the web application to communicate with the API.

Note the API Gateway URL for later use.

URL- https://5h3z7mgl43.execute-api.us-east-2.amazonaws.com/dev

4. Storing Results in DynamoDB:

Create a DynamoDB table named PowerOfMath.

Add an IAM policy to the Lambda function to allow it to write to the DynamoDB table.

Updated Lambda function Code :

```
import json
import math
import boto3
from time import gmtime, strftime

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('PowerOfMath')
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

def lambda_handler(event, context):
    mathResult = math.pow(int(event['base']), int(event['exponent']))
    response = table.put_item(
        Item={
            'ID': str(mathResult),
            'LatestGreetingTime': now
        }
    )
    return {
        'statusCode': 200,
        'body': json.dumps('Your result is ' + str(mathResult))
    }
```

5. Handling Permissions with IAM:

Create an IAM role with permissions to write to DynamoDB and execute Lambda functions.

Attach the following policy to the Lambda function role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:Scan",
        "dynamodb:Query",
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east 2:870114608584:table/PowerOfMath"
    }
  ]
}
```

Connecting AWS Amplify and Lambda -

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>To the Power of Math!</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 base and exponent numbers as parameters
        var callAPI = (base, exponent) => {
            // 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({ "base": base, "exponent": exponent });
            // 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("https://5h3z7mgl43.execute-api.us-east-2.amazonaws.com/dev", requestOptions)
                .then(response => response.text())
                .then(result => alert(JSON.parse(result).body))
                .catch(error => console.log('error', error));
        }
    </script>
</head>
<body>
    <h1>TO THE POWER OF MATH!</h1>
    <form>
        <label>Base number:</label>
        <input type="text" id="base">
        <label>...to the power of:</label>
        <input type="text" id="exponent">
        <!-- set button onClick method to call function we defined passing input values as
parameters -->
        <button type="button" onclick="callAPI(document.getElementById('base').value,
document.getElementById('exponent').value)">CALCULATE</button>
    </form>
</body>
</html>
```

Explanation of the code -

The HTML document begins with basic structure and meta information, including character set and title. The body contains a form with two input fields for entering a base number and an exponent, along with a button labeled "**CALCULATE**". The button triggers a JavaScript function to process the input values. The CSS styling section formats various elements on the page: the heading (h1) is styled with white text, a specific font, and left margin; the body has a dark background; labels have green text, a specific font and size, and margins; buttons have a green background, white text, bold font, and defined dimensions; and input fields have dark text, a specific font, and margins.

The JavaScript function, **callAPI**, is designed to handle the calculation request. It takes the base and exponent values from the user input, creates a JSON string from these values, and sets up an HTTP POST request with appropriate headers and body. This request is sent to an API endpoint using the fetch function, which processes the response using promises. If the API call is successful, the result is parsed and displayed in an alert box; if there's an error, it is logged to the console. When the user fills out the input fields and clicks the "CALCULATE" button, the callAPI function sends the input values to the API endpoint, retrieves the calculation result, and displays it to the user.