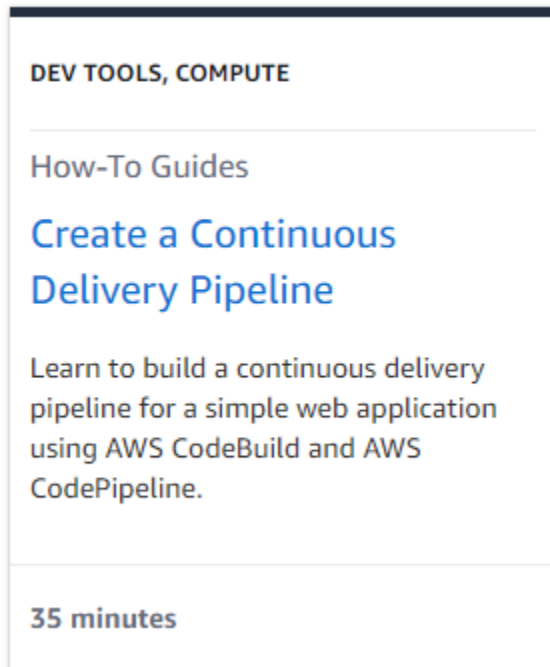


Loc Nguyen

CS.4650.02

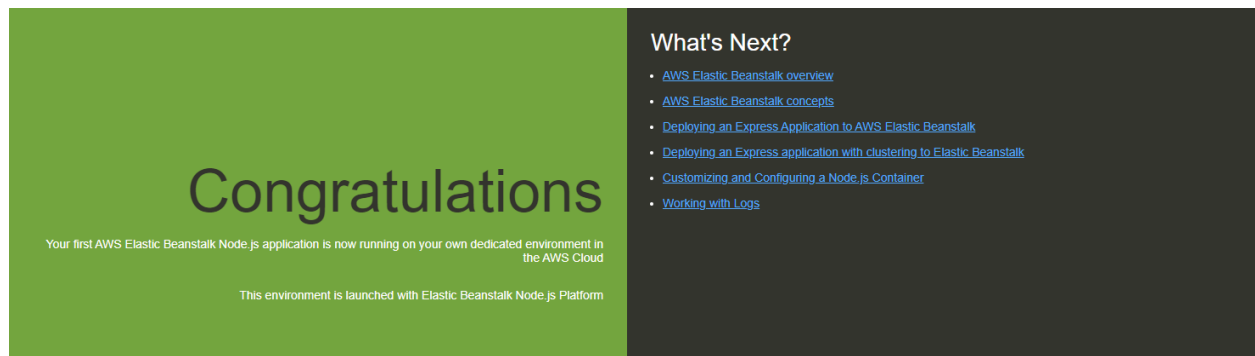
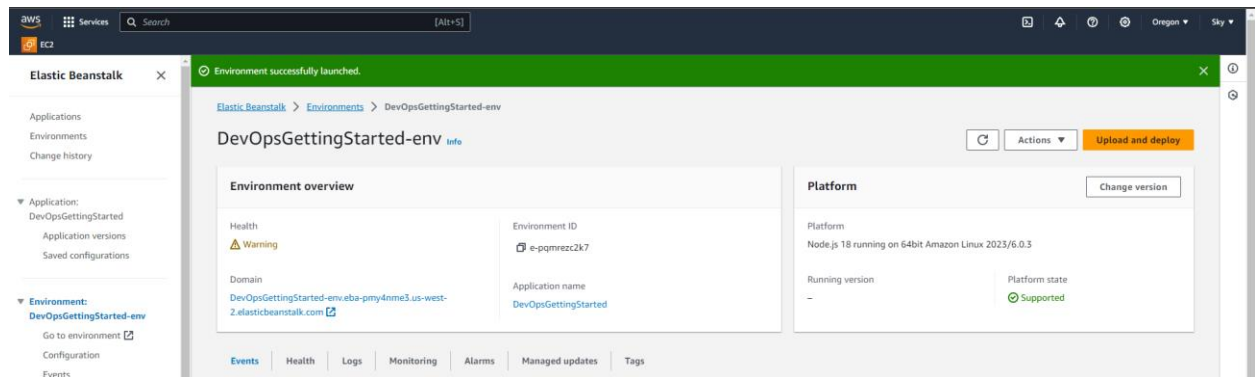
Professor Johanssen

Tutorial 1 – Setting up Continuous Delivery Pipeline

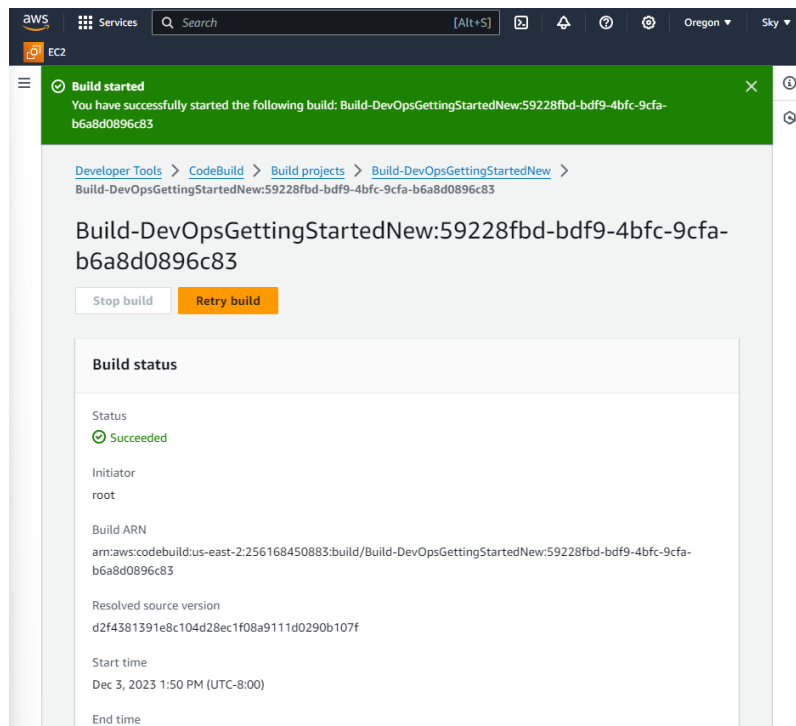


Background – Currently, I’m working on a web application that deploys continuously from GitHub. However, that deployment is with GitHub workflows and is only 1-way, from GitHub repo to AWS EC2 instance. Users data will be written onto the live server database but that information doesn’t get written back to the GitHub database repo file. Hence, every time new patches are pushed, the server database which contains data will get overwritten by the GitHub empty database file. This is a good opportunity for me to learn proper CI/CD pipeline.

1. Launching a web-app with NodeJS using AWS Elastic Beanstalk



2. Using CodeBuild for continuous integration using GitHub as source repo



3. Use CodePipeline for continuous deployment with GitHub as source repo and CodeBuild for auto build and testing

The screenshot displays the AWS CodePipeline console interface. On the left, a navigation pane shows the 'CodePipeline' section expanded, with sub-items like 'Getting started', 'Pipelines', 'Pipeline', 'History', 'Settings', and 'Settings'. The main content area shows the details for a pipeline named 'Pipeline-DevOpsGettingStarted'. The pipeline type is 'V2'. The execution ID is '461b0774-3cfd-4f66-b844-d375236fe5e1'. The pipeline consists of three stages: 'Source', 'Build', and 'Deploy'. Each stage is marked as 'Succeeded'.

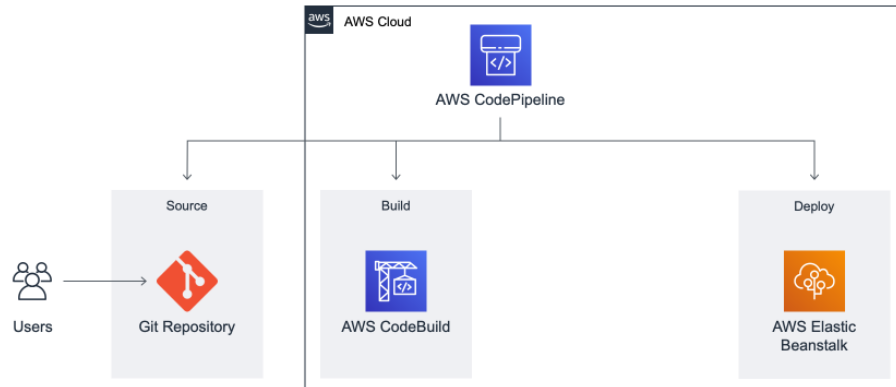
Source Stage: Succeeded. Pipeline execution ID: 461b0774-3cfd-4f66-b844-d375236fe5e1. Source: GitHub (Version 1). Succeeded - 2 minutes ago. d2f43813. Source: change message.

Build Stage: Succeeded. Pipeline execution ID: 461b0774-3cfd-4f66-b844-d375236fe5e1. Build: AWS CodeBuild. Succeeded - 1 minute ago. Details. View logs. d2f43813. Source: change message.

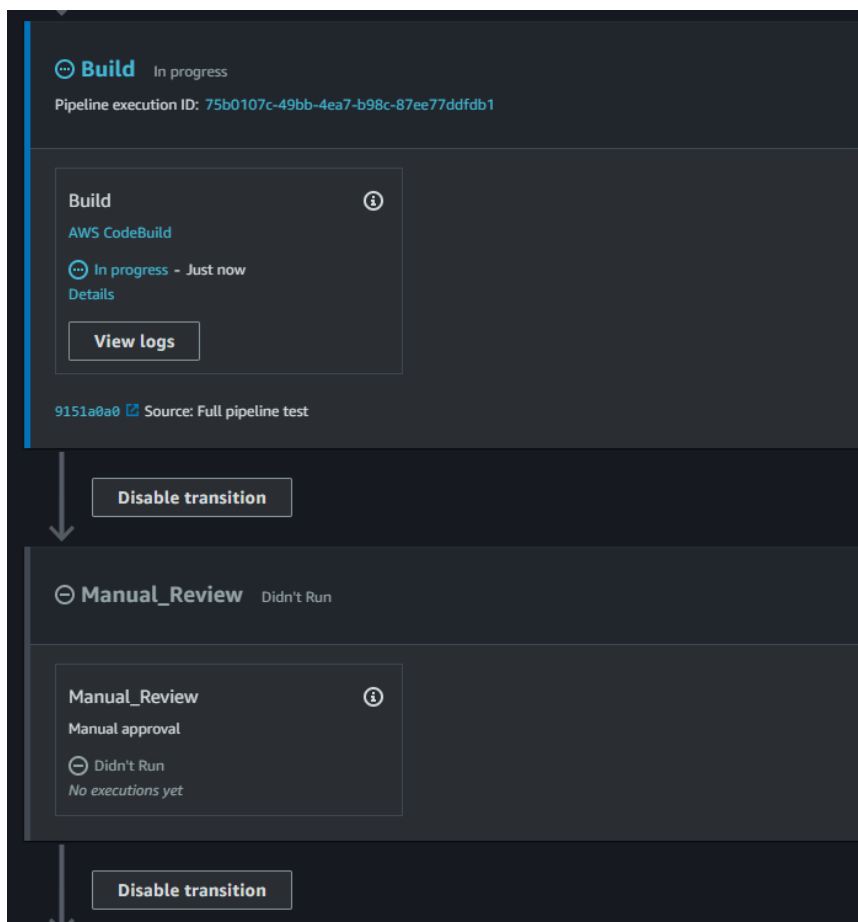
Deploy Stage: Succeeded. Pipeline execution ID: 461b0774-3cfd-4f66-b844-d375236fe5e1. Deploy: AWS Elastic Beanstalk. Succeeded - Just now. d2f43813. Source: change message.

Application architecture

Here's what our architecture looks like now:



4. Adding manual review as a step to code deployment pipeline. Print out a line “I love blue” to test the pipeline

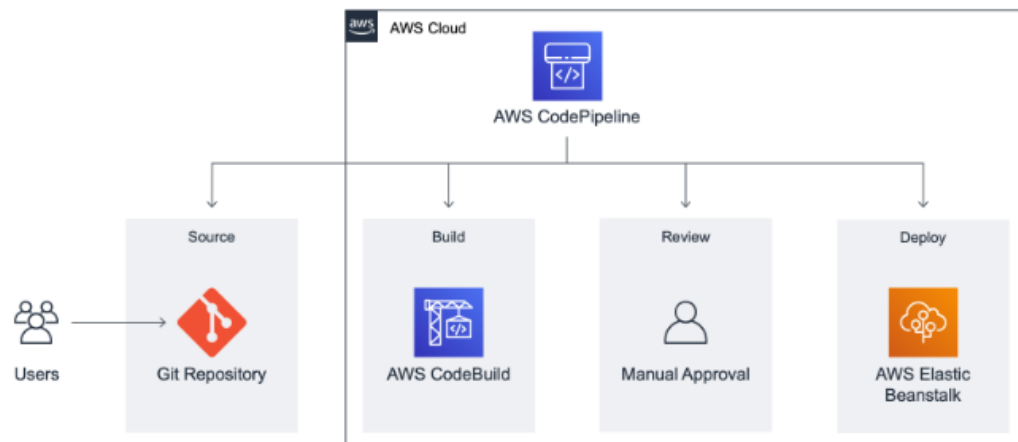


I love blue

5. Winding down and deleting resources
6. Final architecture

Application architecture

With all modules now completed, here is the architecture of what you built:



I'm super proud that I know a little bit more about how to deploy code secured and fast with AWS microservices. I do want to learn more about domain registration and firewall policies when deploying a production websites versus a development website.

Tutorial 2 – Build a Basic Web Application Static and serverless

SERVERLESS**UPDATED**

Tutorial

Build a Basic Web Application

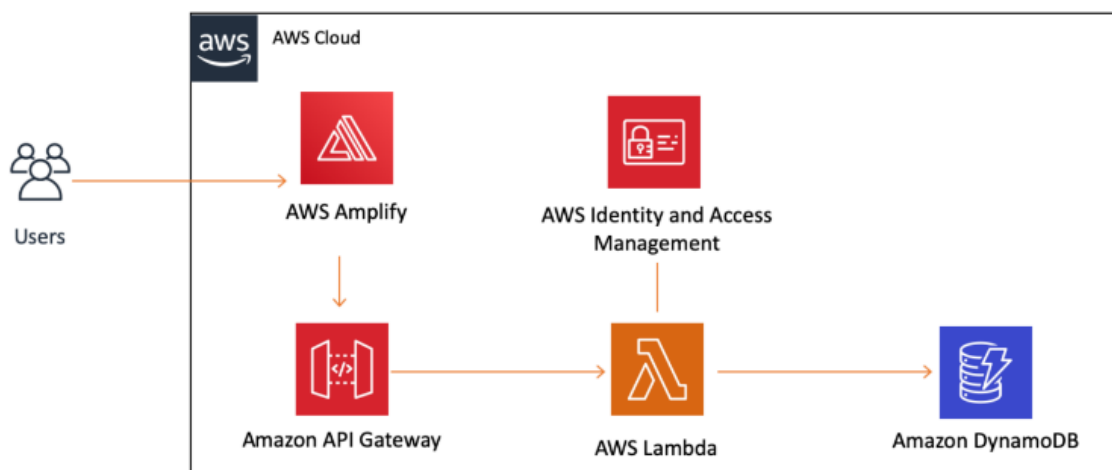
Build a static web app that renders "Hello World." Then, learn how to add functionality to the web app so the text that displays is based on a custom input you provide.

<

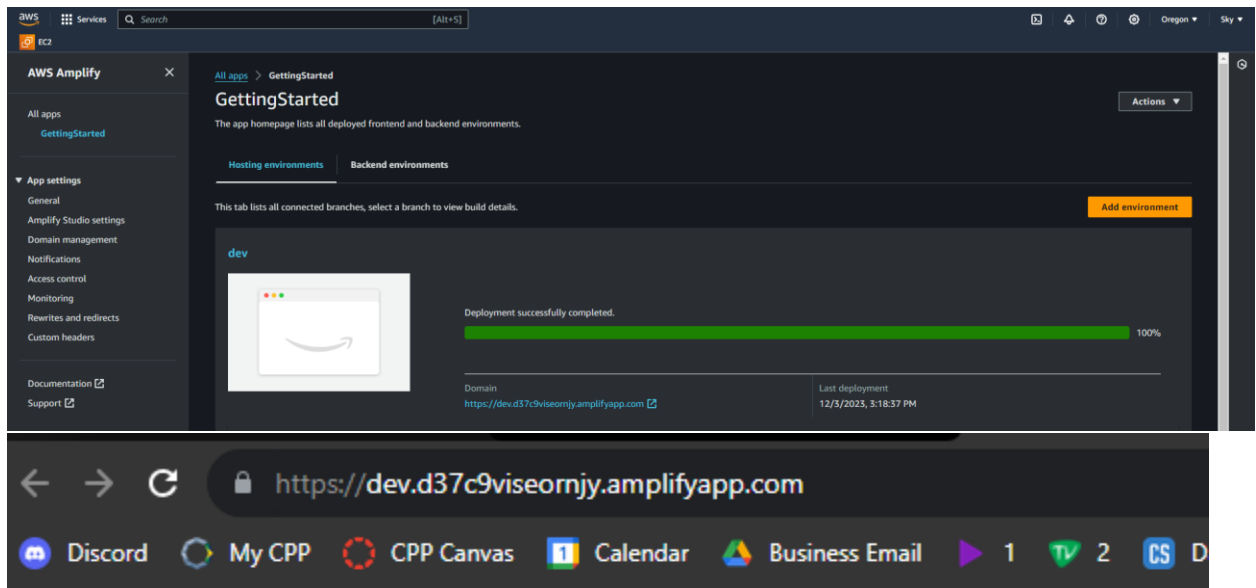
30 minutes

Background – Web application are great for new businesses. I'm excited to learn more about different ways to create web app

1. Overall architecture

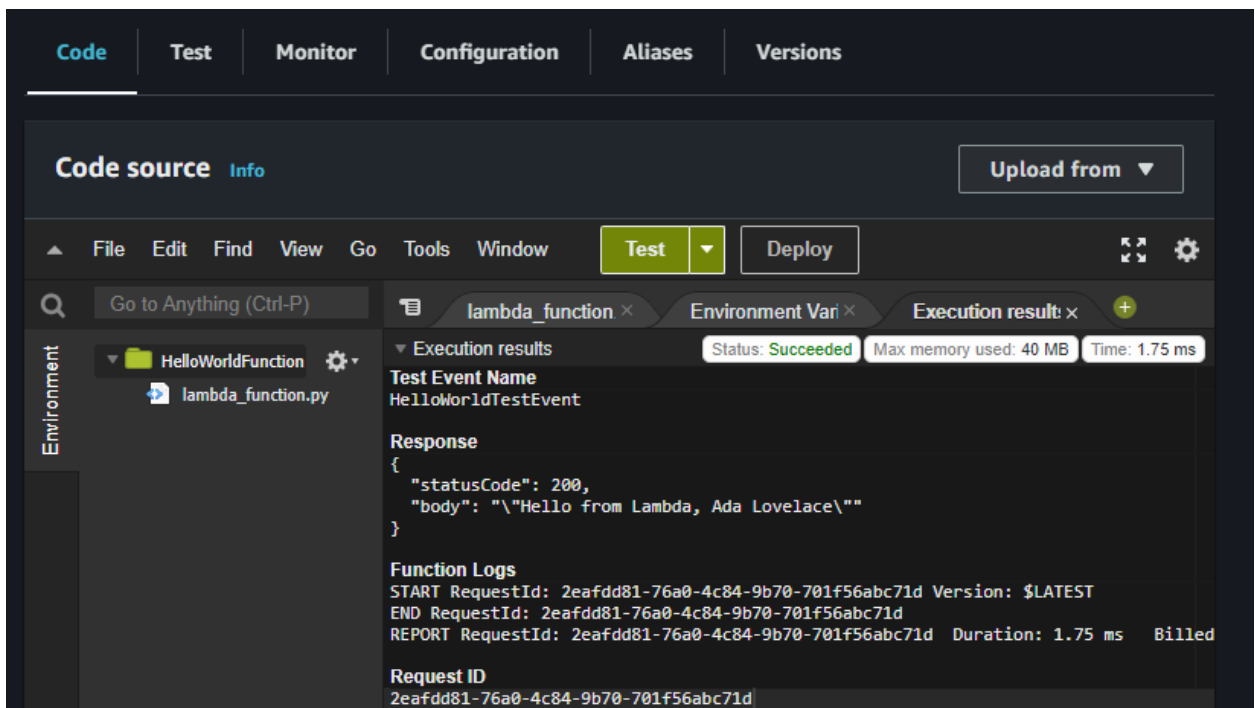
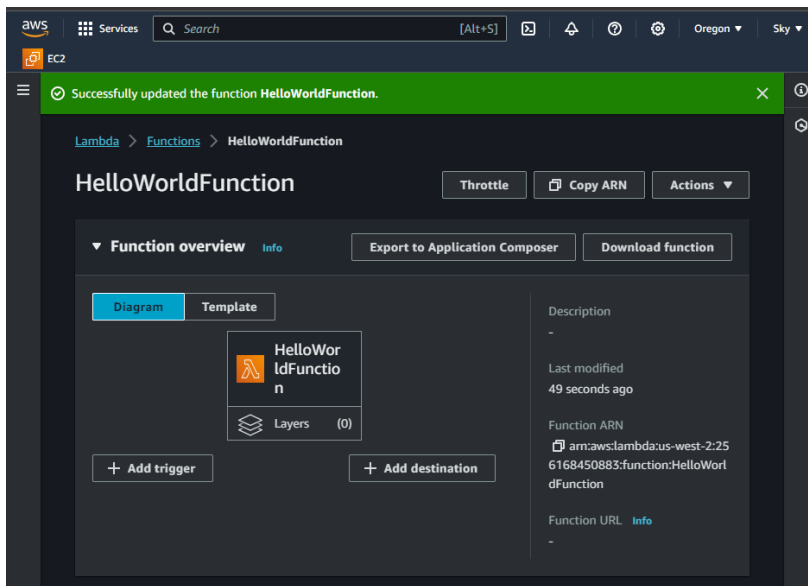


2. Deploy the web app with zip file index.html with AWS Amplify



Hello World

3. Create serverless functions with AWS Lambda and test it



4. Use Amazon API Gateway to create RESTful API to make calls to Lambda function from web client. Successfully deployed an API and tested it. Response with code 200 OK.


Create method


Method details


Method type


POST ▼


Integration type

☒ **Lambda function**
Integrate your API with a Lambda function.


☐ **HTTP**
Integrate with an existing HTTP endpoint.


☐ **Mock**
Generate a response based on API Gateway mappings and transformations.


☐ **AWS service**
Integrate with an AWS Service.


☐ **VPC link**
Integrate with a resource that isn't accessible over the public internet.


☐ **Lambda proxy integration**
Send the request to your Lambda function as a structured event.

Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-west-2 ▼ 🔍 `arn:aws:lambda:us-west-2:256168450883:function:He` ✕

Request body

```
1 {
2   "firstName": "Grace",
3   "lastName": "Hopper"
4 }
```

Test

POST method test results

Request	Latency	Status
/	441	200

Response body

```
{
  "statusCode": 200,
  "body": "Hello from Lambda, Grace Hopper!"
}
```

5. Use DynamoDB to store data. Use AWS IAM to give services permissions to interact with each other.

Created a DynomoDB table and IAM policy

≡

[IAM](#) > [Roles](#) > [HelloWorldFunction-role-direpf0p](#) > [Create policy](#)

Step 1
[Specify permissions](#)

Step 2
Review and create

Review and create Info

Review the permissions, specify details, and tags.

Policy details

Policy name
Enter a meaningful name to identify this policy.

HelloWorldDynamoPolicy

Maximum 128 characters. Use alphanumeric and '+=, @-_' characters.

Permissions defined in this policy Info

Edit

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Q Search

Allow (1 of 402 services)

☒ Show remaining 401 services

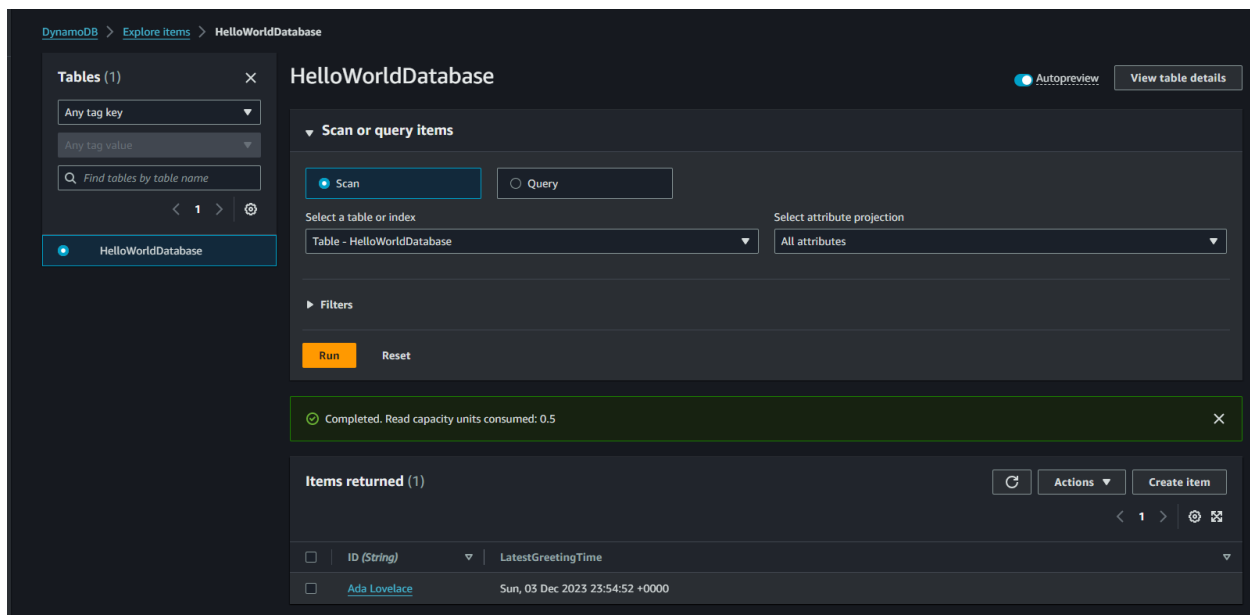
Service	Access level
DynamoDB	Limited: Read, Write

Cancel

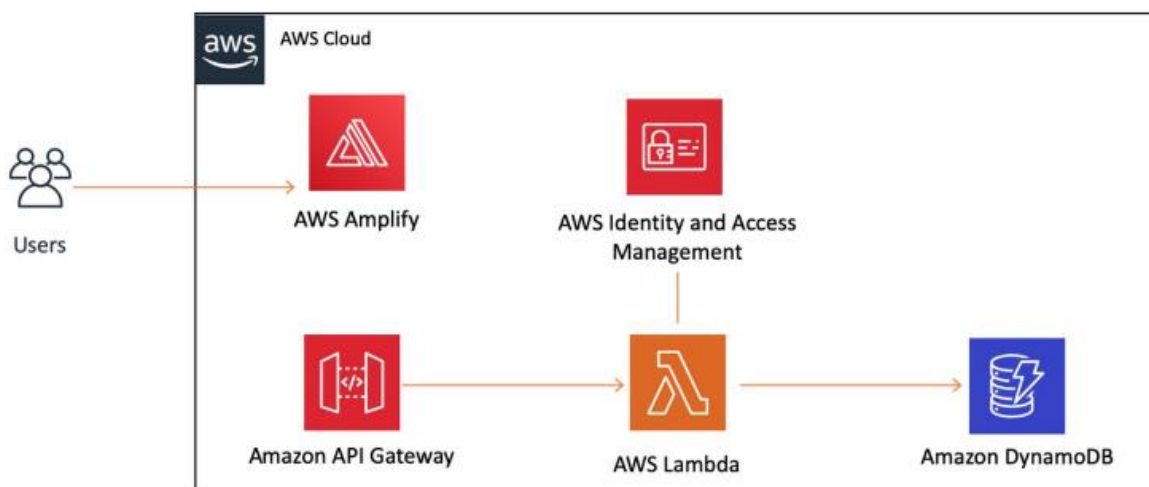
Previous

Create policy

Tested and made sure data is written into the DB



Current Architecture



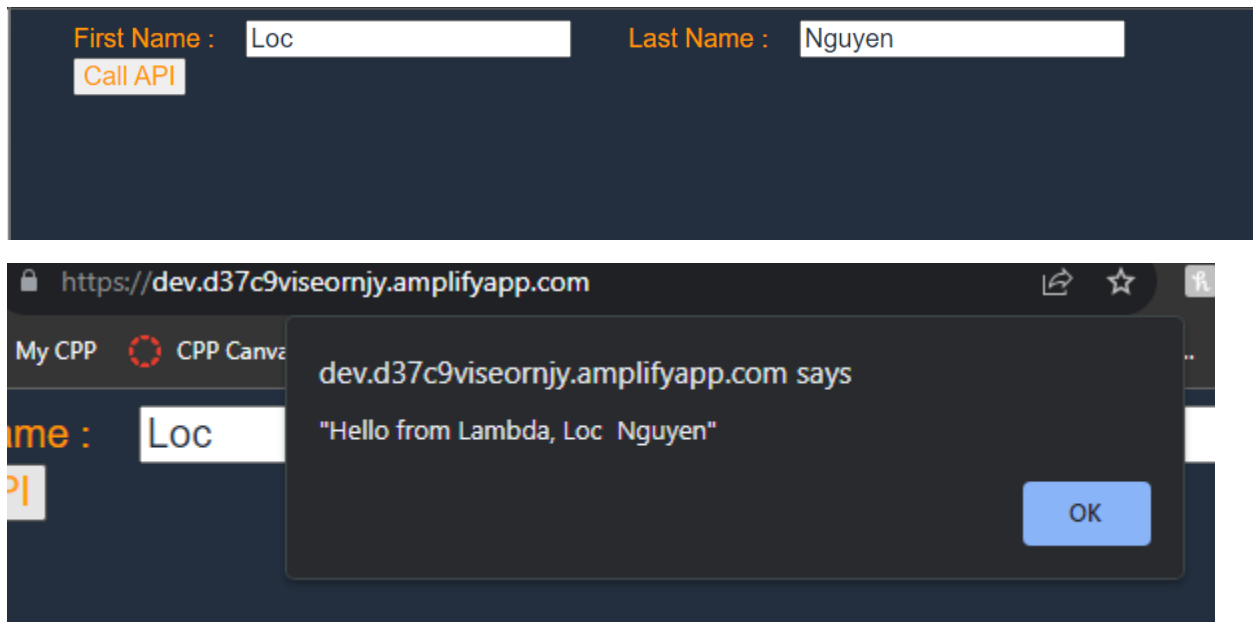
6. Call an API Gateway from an HTML image

Modify the original HTML to call the API Gateway using the invoke URL -

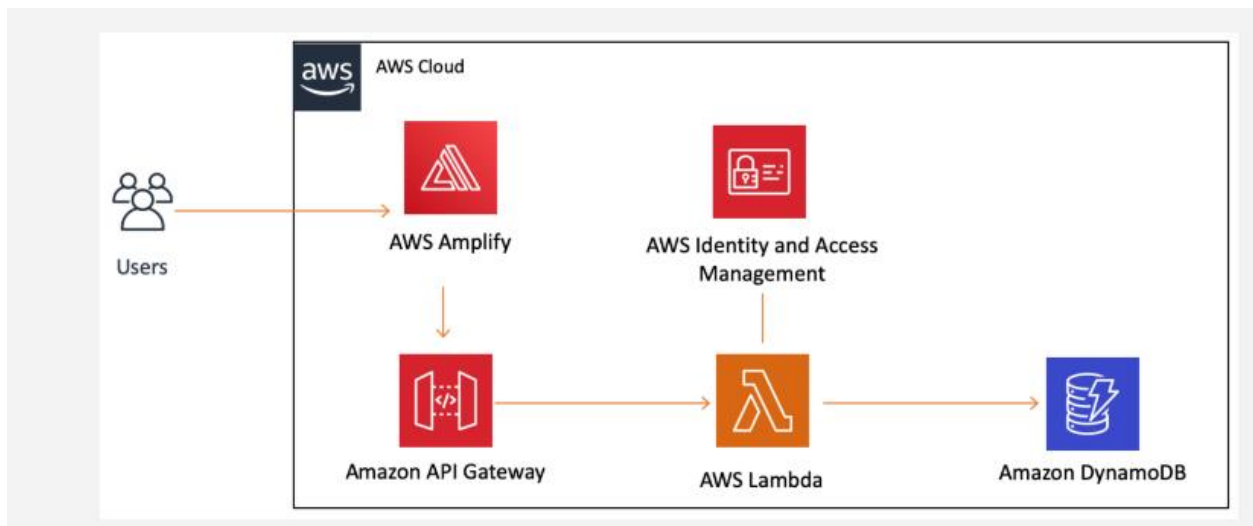
<https://2jl7k63sh2.execute-api.us-east-2.amazonaws.com/dev> and deploy it on AWS Amplify

```
    redirect: 'follow'
  });
  // make API call with parameters and use promises to get response
  fetch("https://2jl7k63sh2.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));
```

Tested the interactive website with an API call



Final application architecture



End of lesson takeaway – super stoked to learn how web applications deployment work and will incorporate my learning into future projects. I'm curious about limitations behind this process and thinking about how I can utilize this to make a weather app calling the OpenWeather API that's has 1000 free API call everyday.