# AWS Lambda

## **Introduction**

Let’s build a simple serverless application using AWS Lambda.

This application will have an html front end hosted on AWS Amplify, where you can enter some text. On submitting the form, it will provide you with a response which is capitalized and reverse of your entered text.

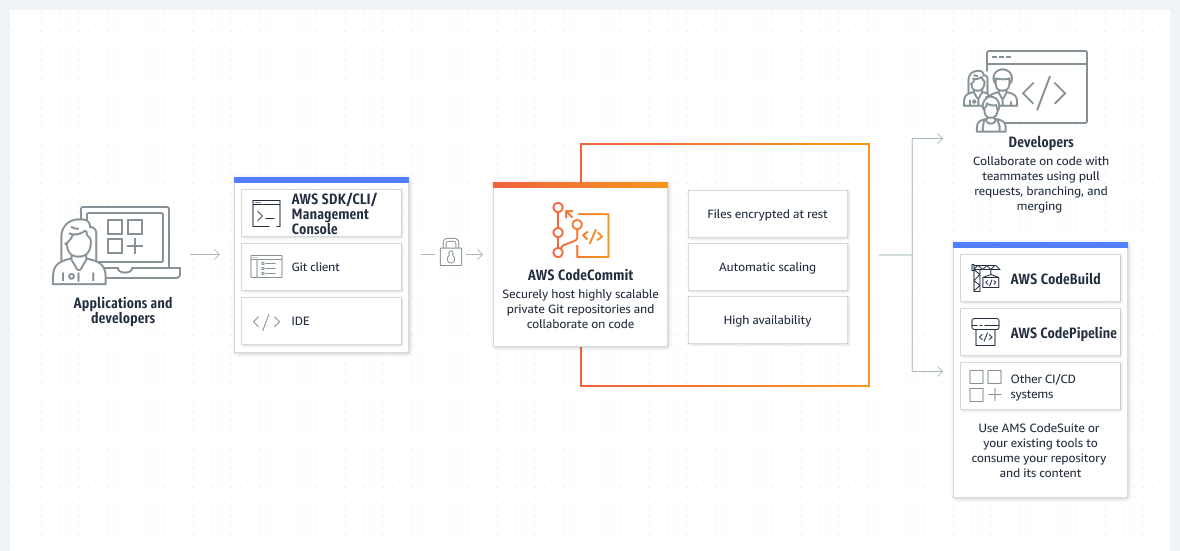
Capitalize and Reverse will be two separate Lambda functions to show you the chaining capabilities. Instead of accessing these functions directly, an API Gateway will be used to accept client requests and respond with the final output.

The components we use are:

## **AWS CodeCommit**

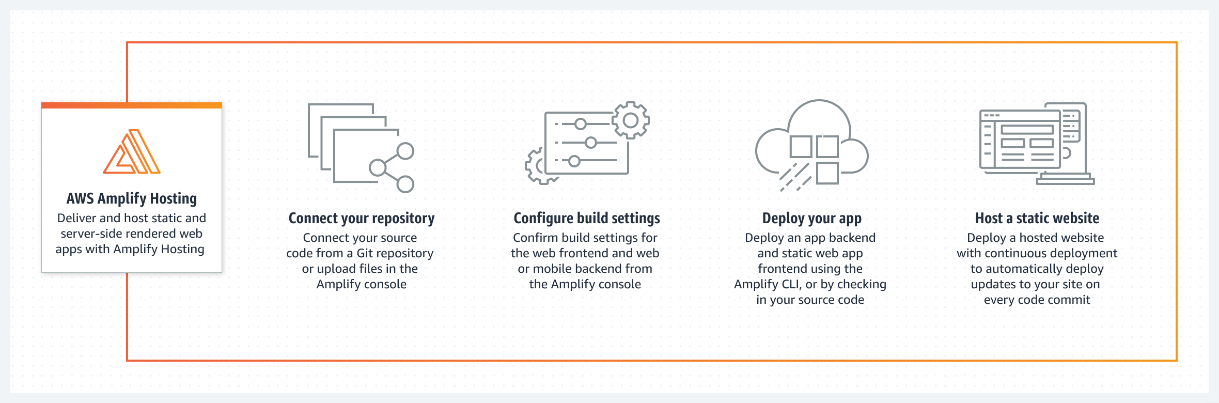
AWS CodeCommit is a secure, highly scalable, fully managed source control service that hosts private Git repositories.

As a Git-based service, CodeCommit is well suited to most version control needs. There are no arbitrary limits on file size, file type, and repository size.



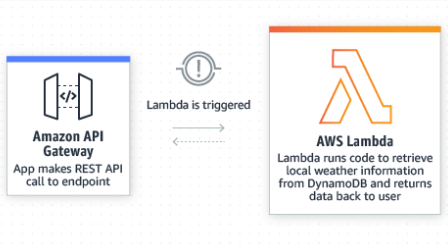
## **AWS Amplify**

AWS Amplify is a complete solution that lets front end web and mobile developers easily build, ship, and host full-stack applications on AWS, with the flexibility to leverage the breadth of AWS services as use cases evolve.



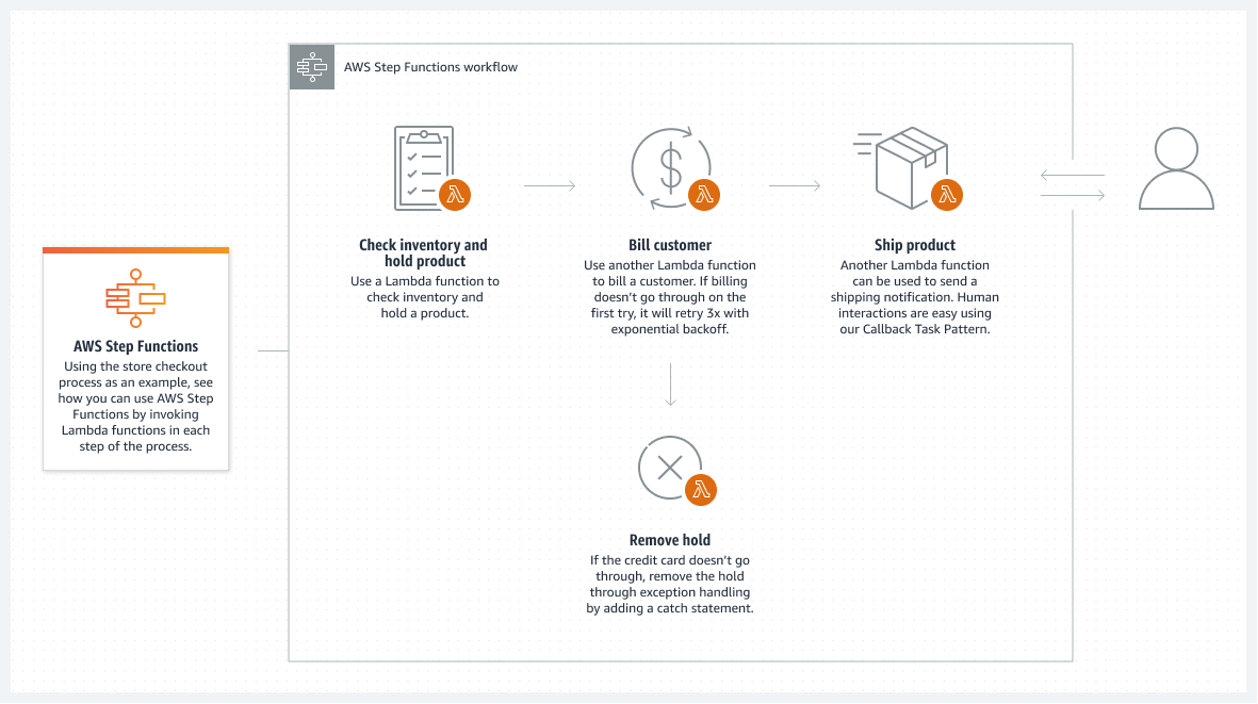
## **AWS Lambda Functions**

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or back end service without provisioning or managing servers. You can trigger Lambda from over 200 AWS services and software as a service (SaaS) application, and only pay for what you use.



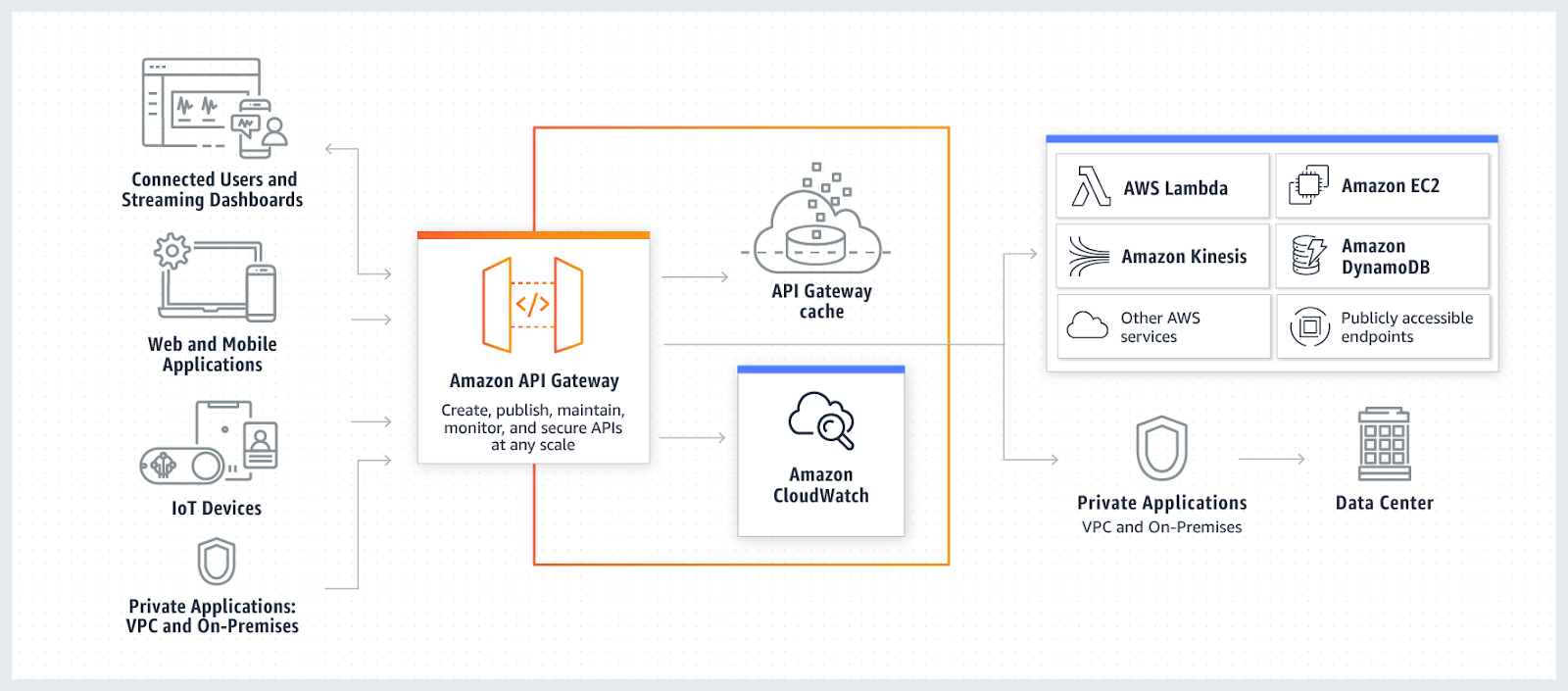
## AWS Step Function

AWS Step Function is a visual workflow service that helps developers use AWS services to build distributed applications, automate processes, orchestrate microservices, and create data and machine learning (ML) pipelines.



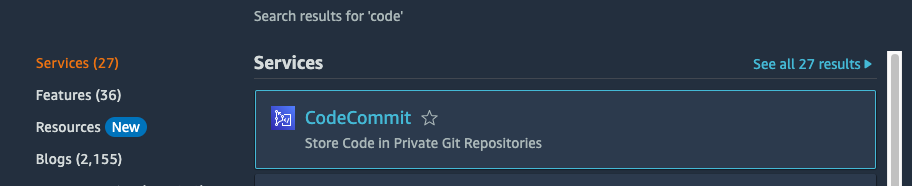
## **AWS API Gateway**

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your back end services.

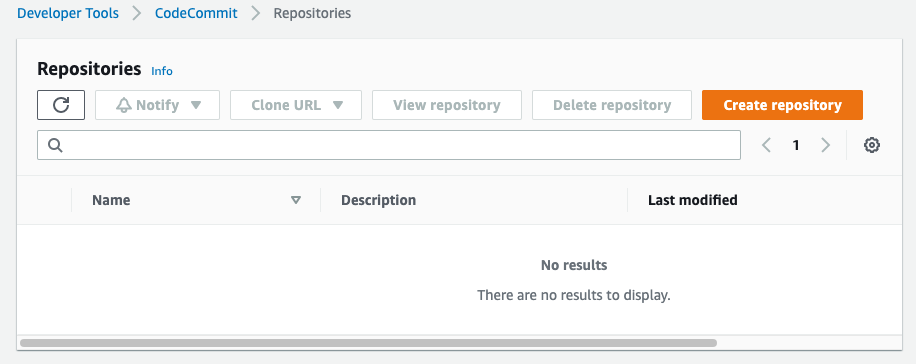


# **Process**

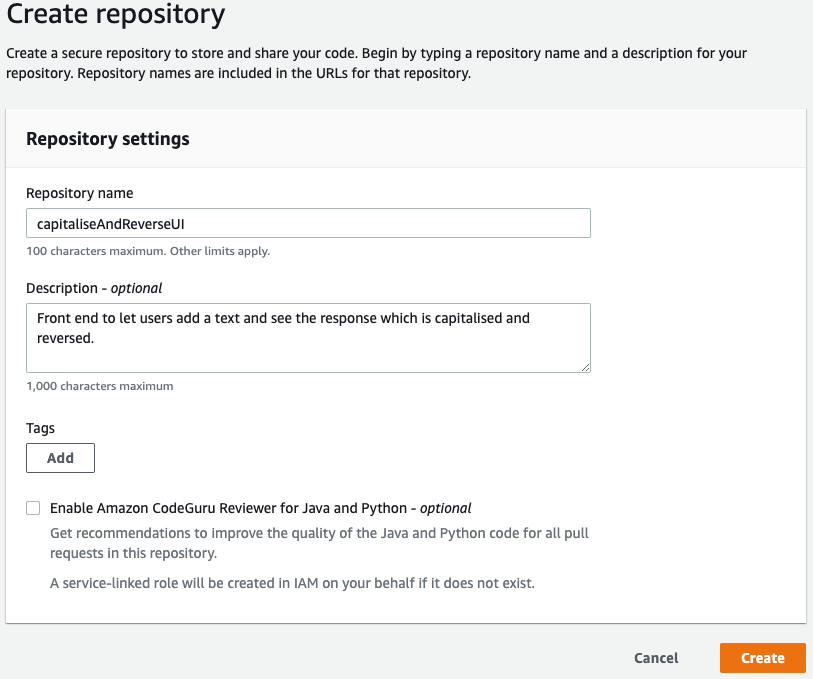
Let’s start with defining CodeCommit resource which you can use as your code repository.



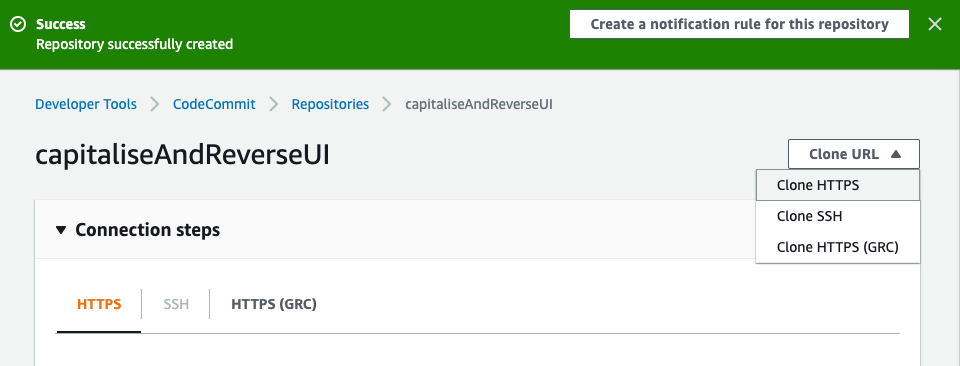
You start with a blank repository. Click on **Create repository**.



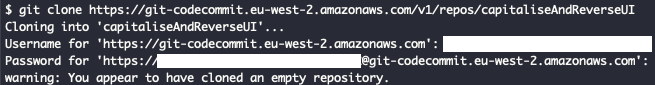
Provide repository name and optional description.



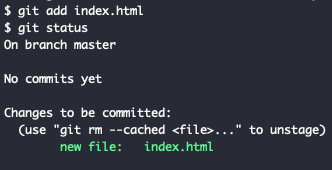
Now get the details of this repository to clone in your local environment.



Clone the repository on your computer to create the required html resources.



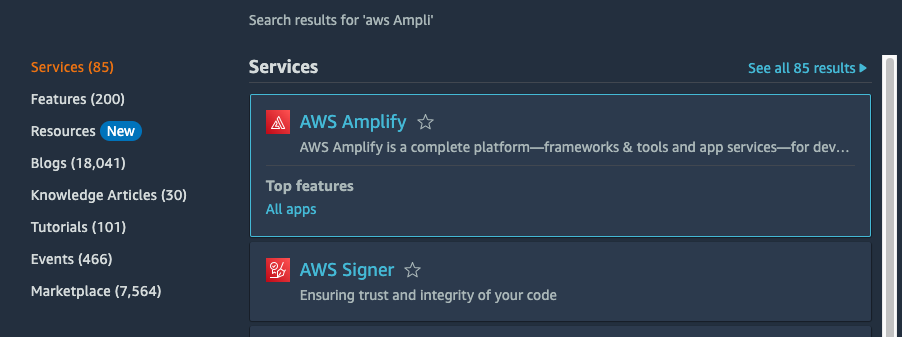
You then create a simple html page (that will contain the require JavaScript and CSS sections).



Commit your changes and you can also push the changes to the remote repository on AWS CodeCommit.



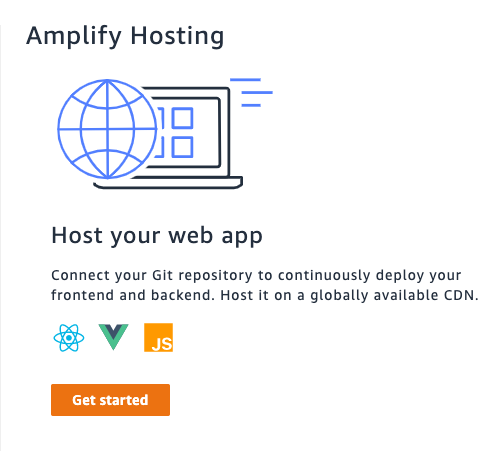
Now create AWS Amplify resource to host your static content (HTML).



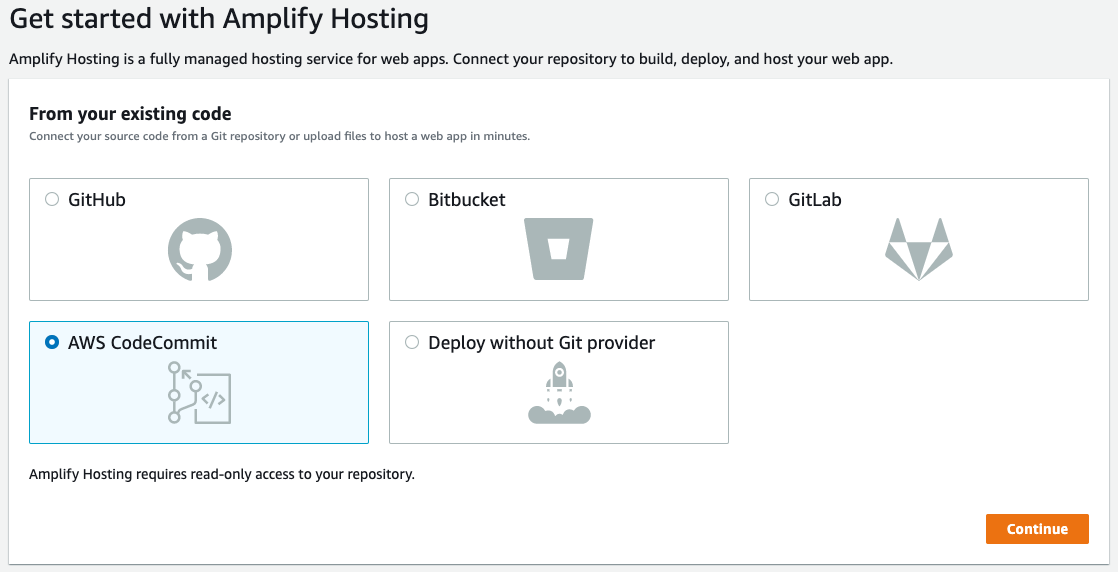
Start by creating the resource.



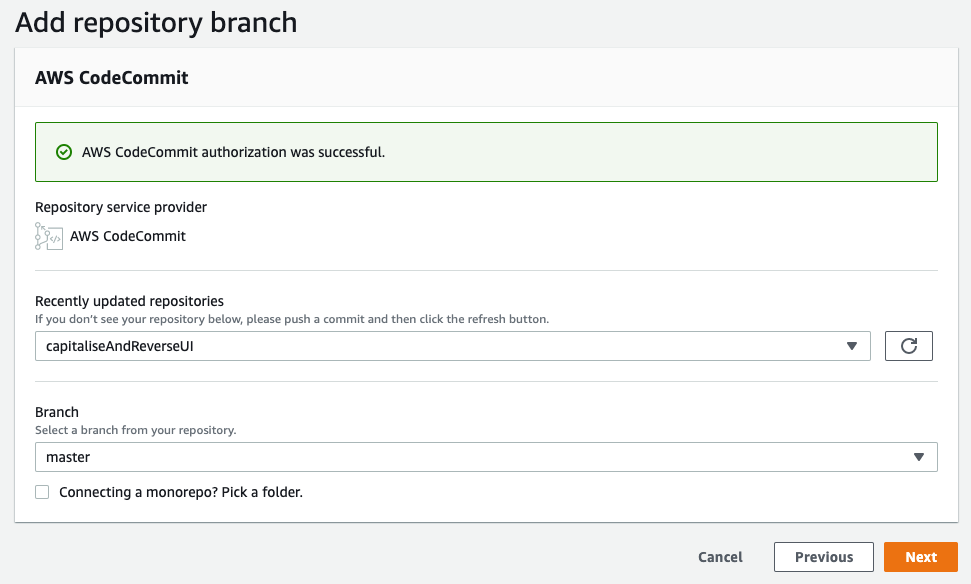
Choose **Host your web app**.



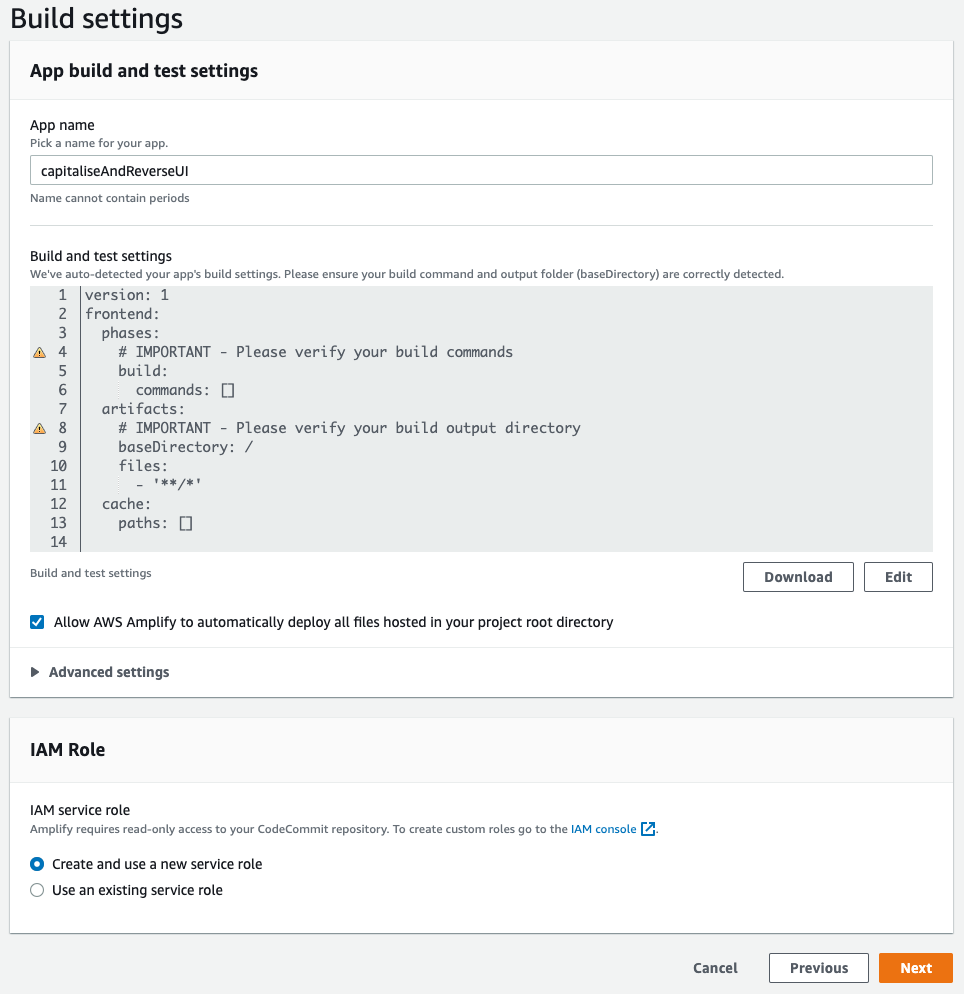
Select AWS CodeCommit; this is where you have pushed changes from your local environment to the repository.



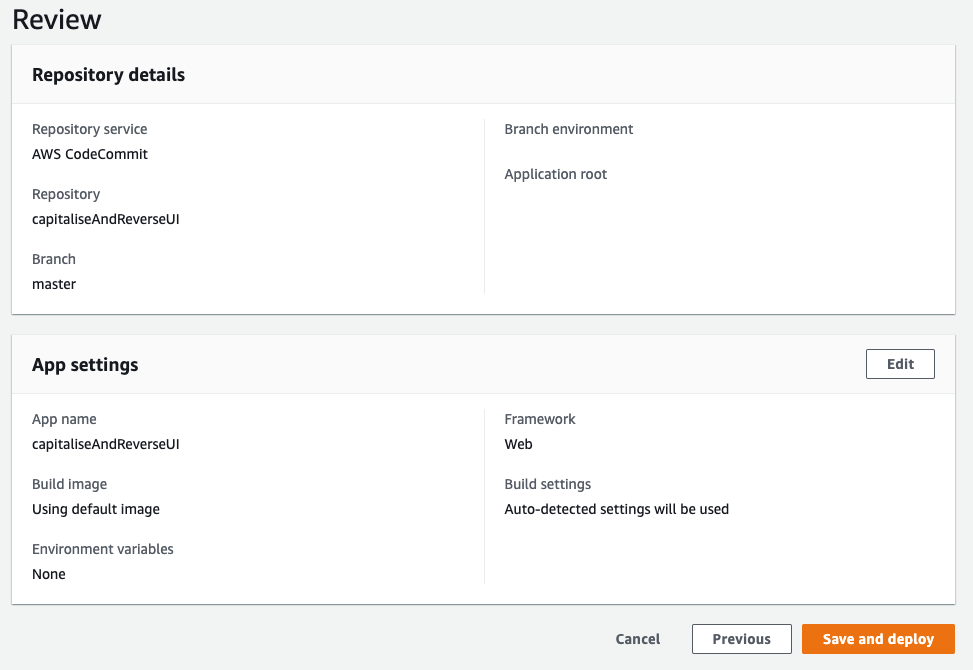
You will now link the master branch with AWS Amplify. This will provide the continuous delivery for you whenever you push changes to master branch.



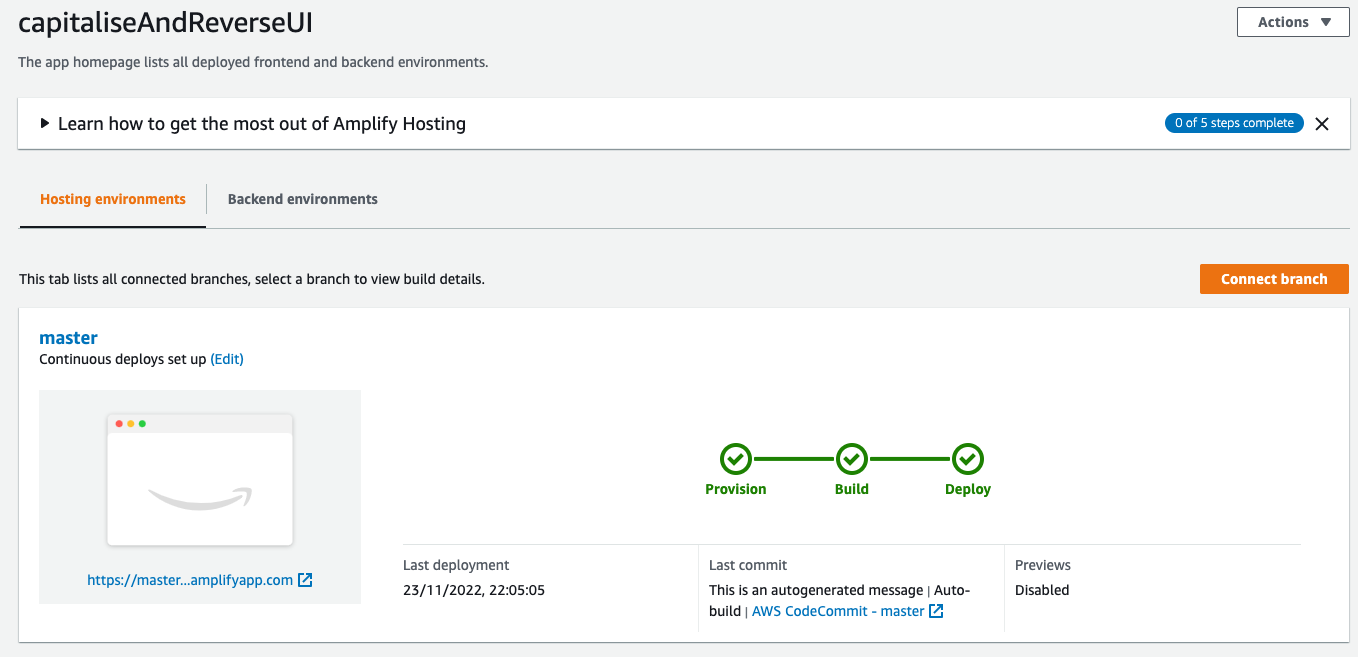
Accept the default build settings.



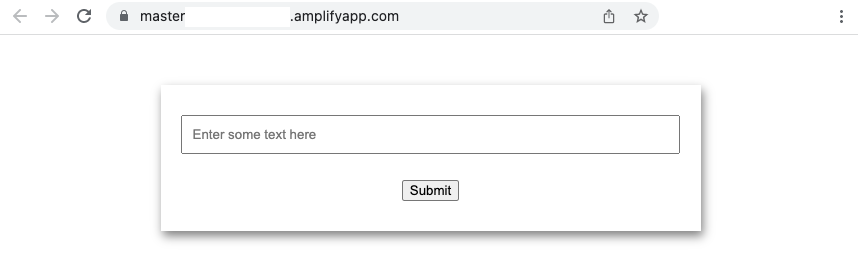
Review and complete the process.



Process takes some time to complete (provisioning, building, and deploying your changes).

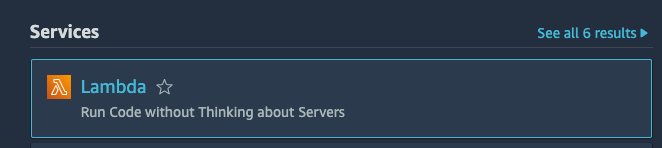


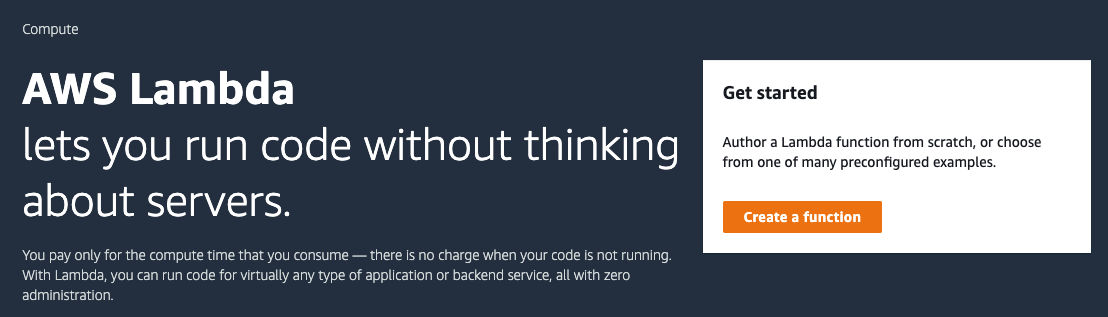
Once completed, you can visit the URL to see your web application in action.



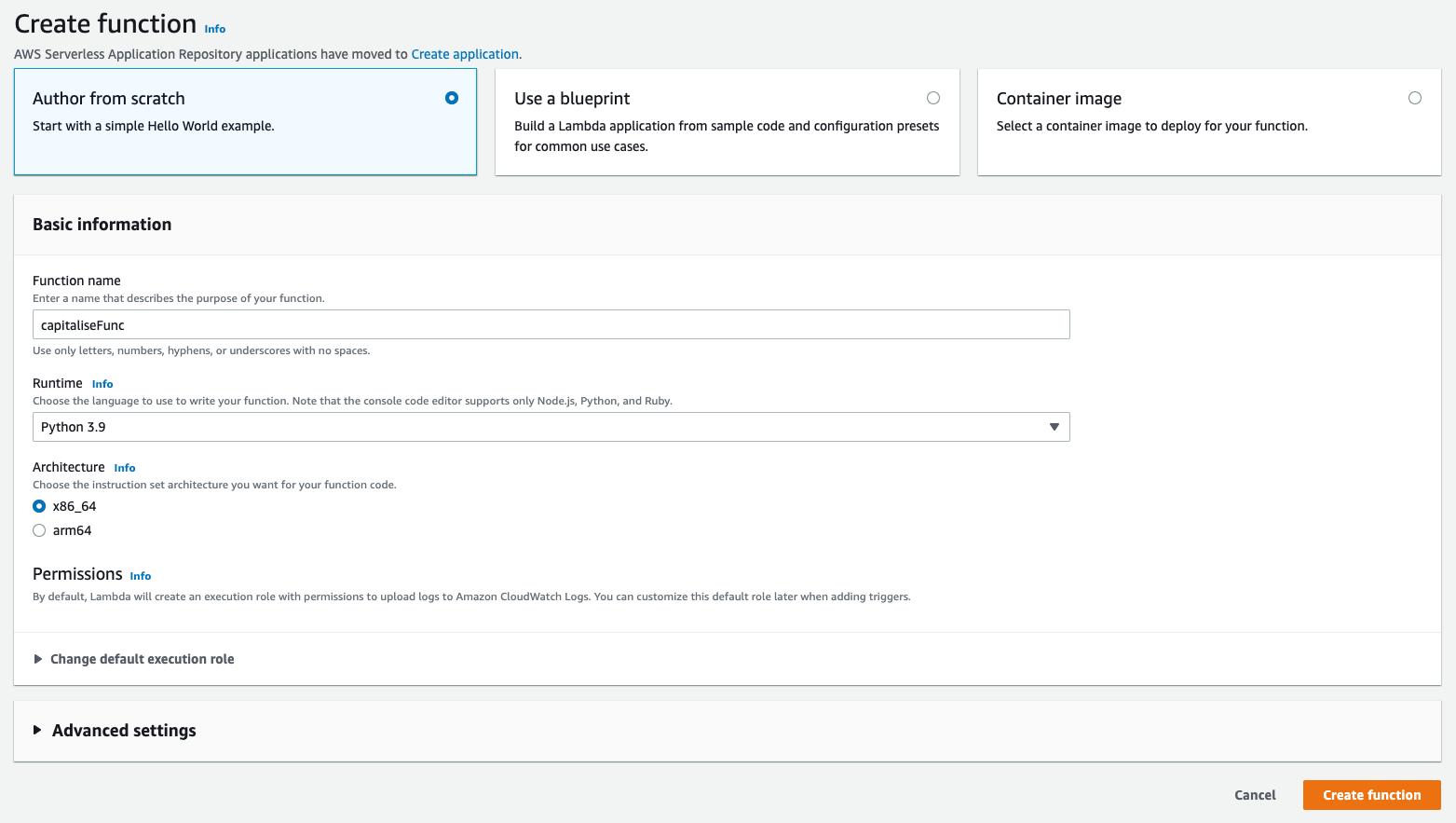
But this application is not complete, you are yet to build the back end to do the capitalization and reversal of the input string.

You start by defining the first AWS Lambda function to Capitalize the input text.

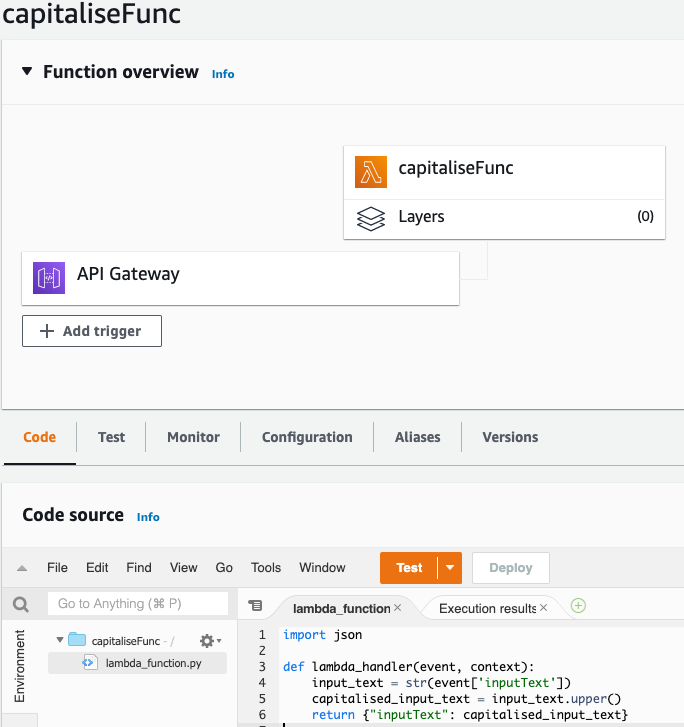




Provide the function name and runtime. Choose Python 3.9 for this.



Defining the function will look like this:



The code you have written is very basic, as it accepts input text as part of the body (it’s a HTTP POST function). And returns the object again as input text with capitalized value (so you can chain this to the reverse function).

import json

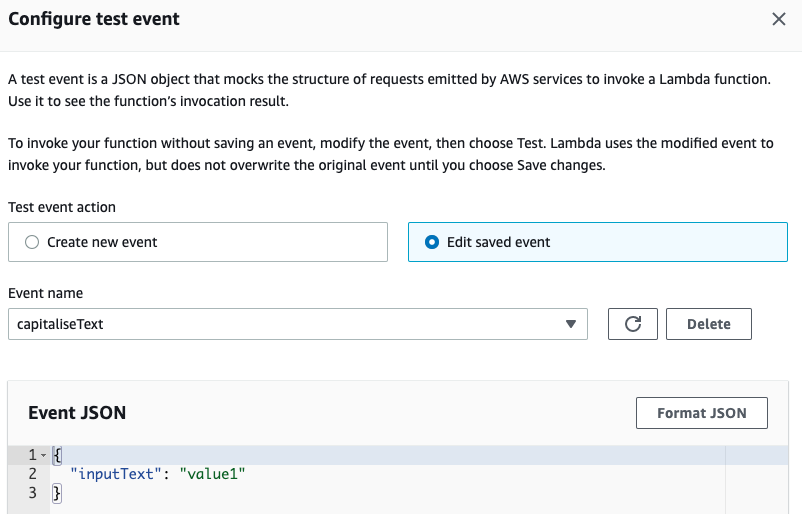
def lambda\_handler(event, context):

input\_text = str(event['inputText'])

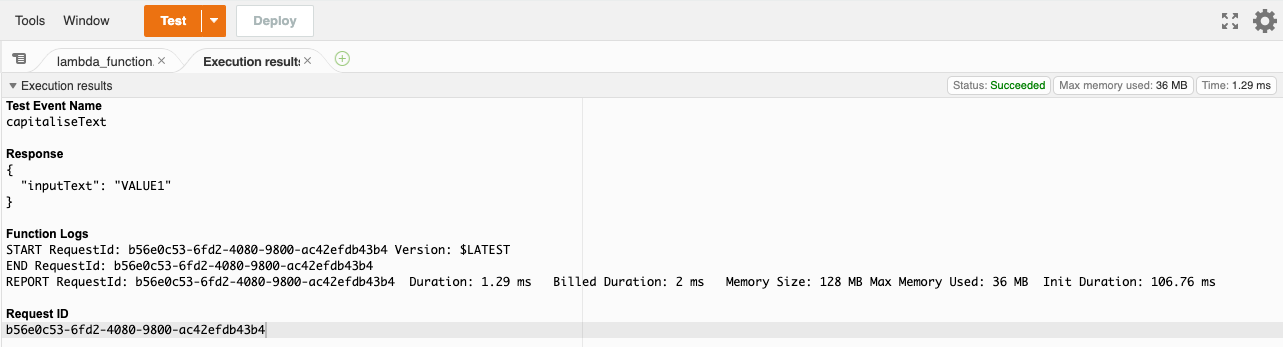
capitalised\_input\_text = input\_text.upper()

return {"inputText": capitalised\_input\_text}

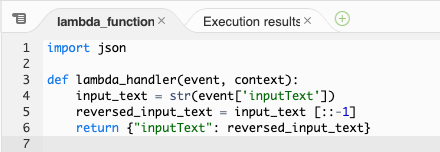
You can also define a simple test event to validate your function logic.



And once you deploy your function, you can then test it and see the following outcome.



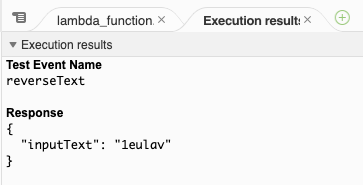
Similarly, you create the reverse function.



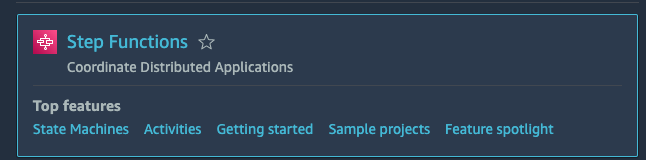
import json

def lambda\_handler(event, context): input\_text = str(event['inputText']) reversed\_input\_text = input\_text [::-1] return {"inputText": reversed\_input\_text}

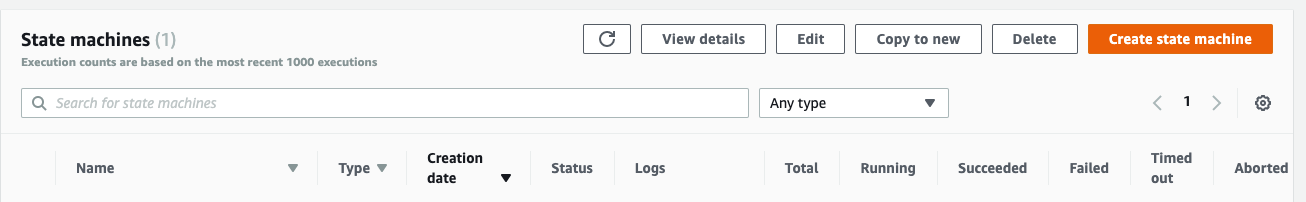
And test it



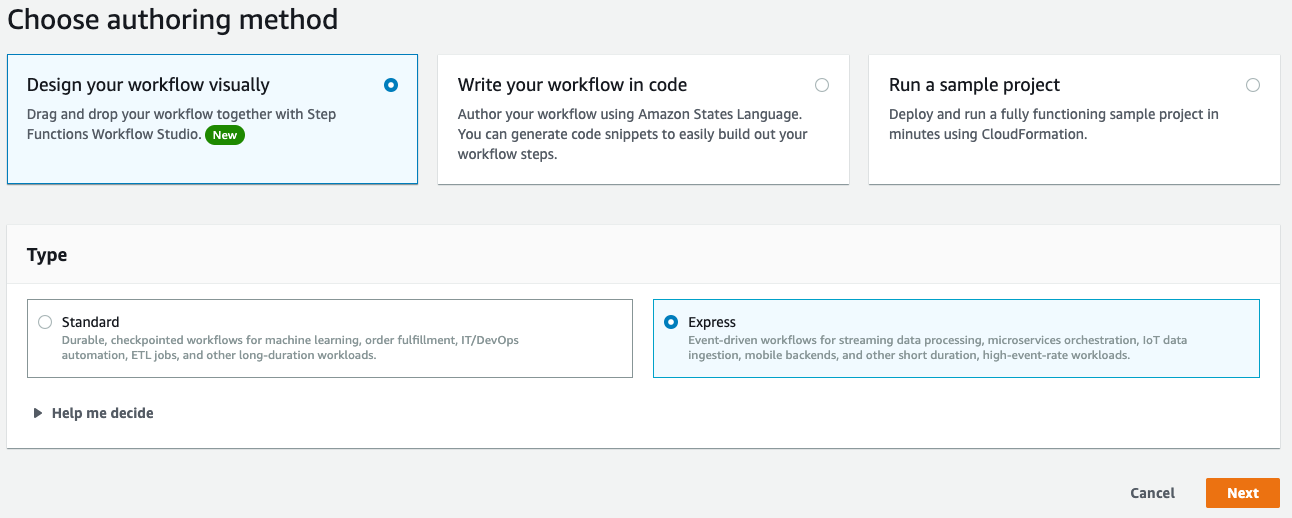
Now that you have two functions defined and created, you can chain them together using StepFunctions.

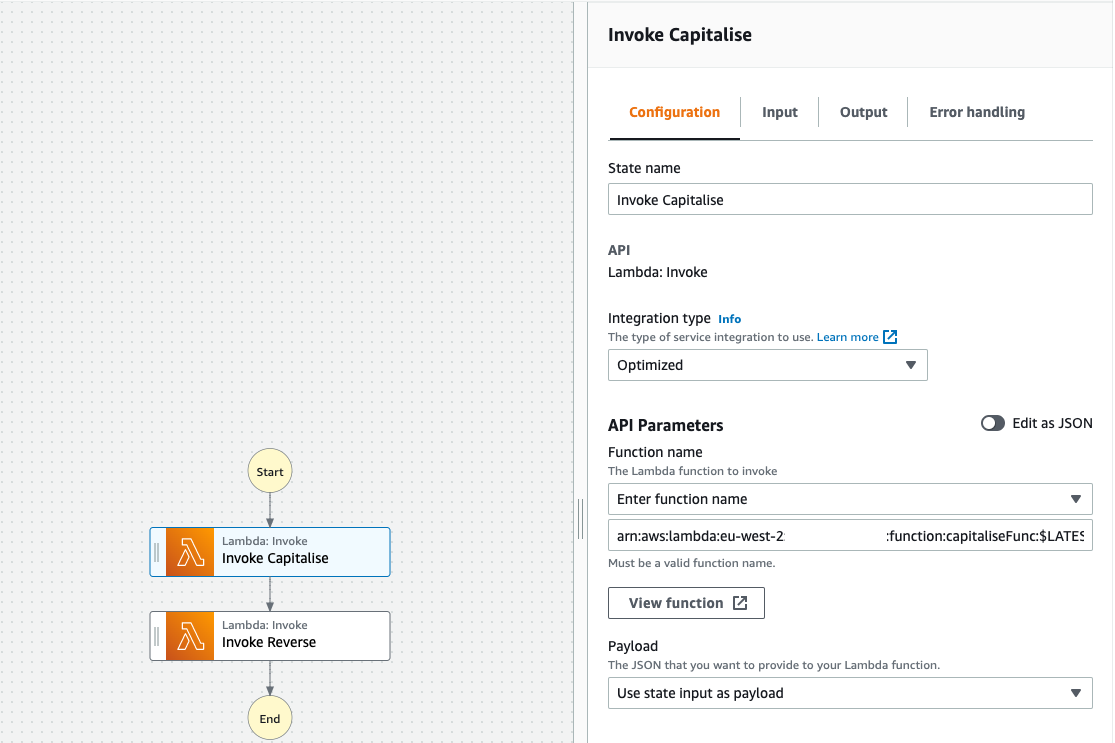


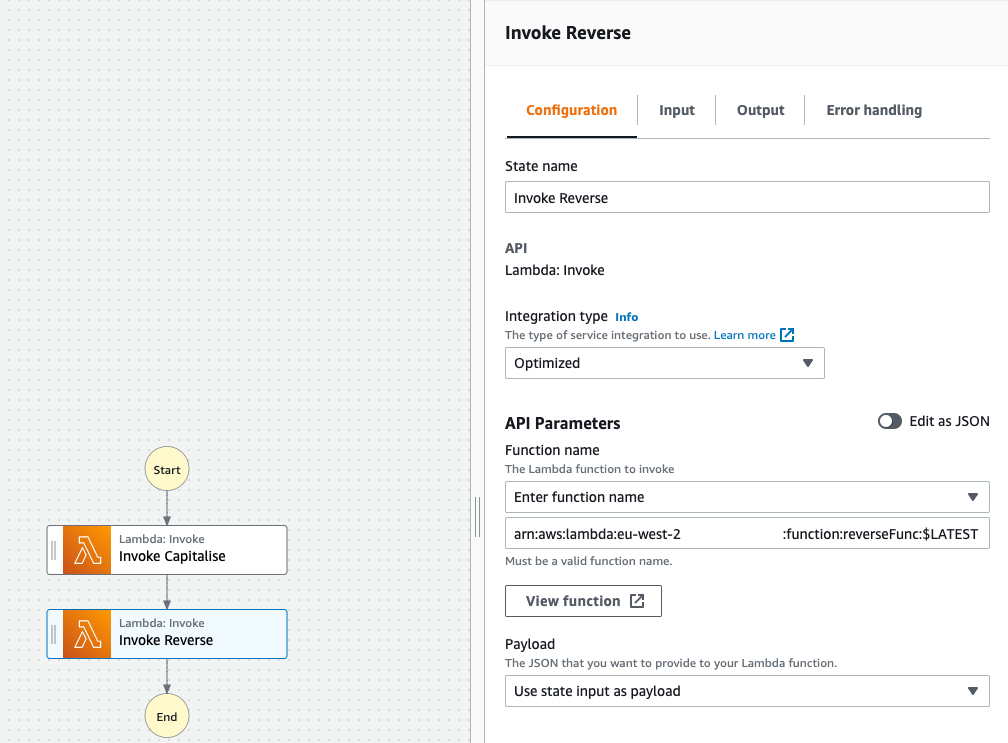
Start by creating a state machine.

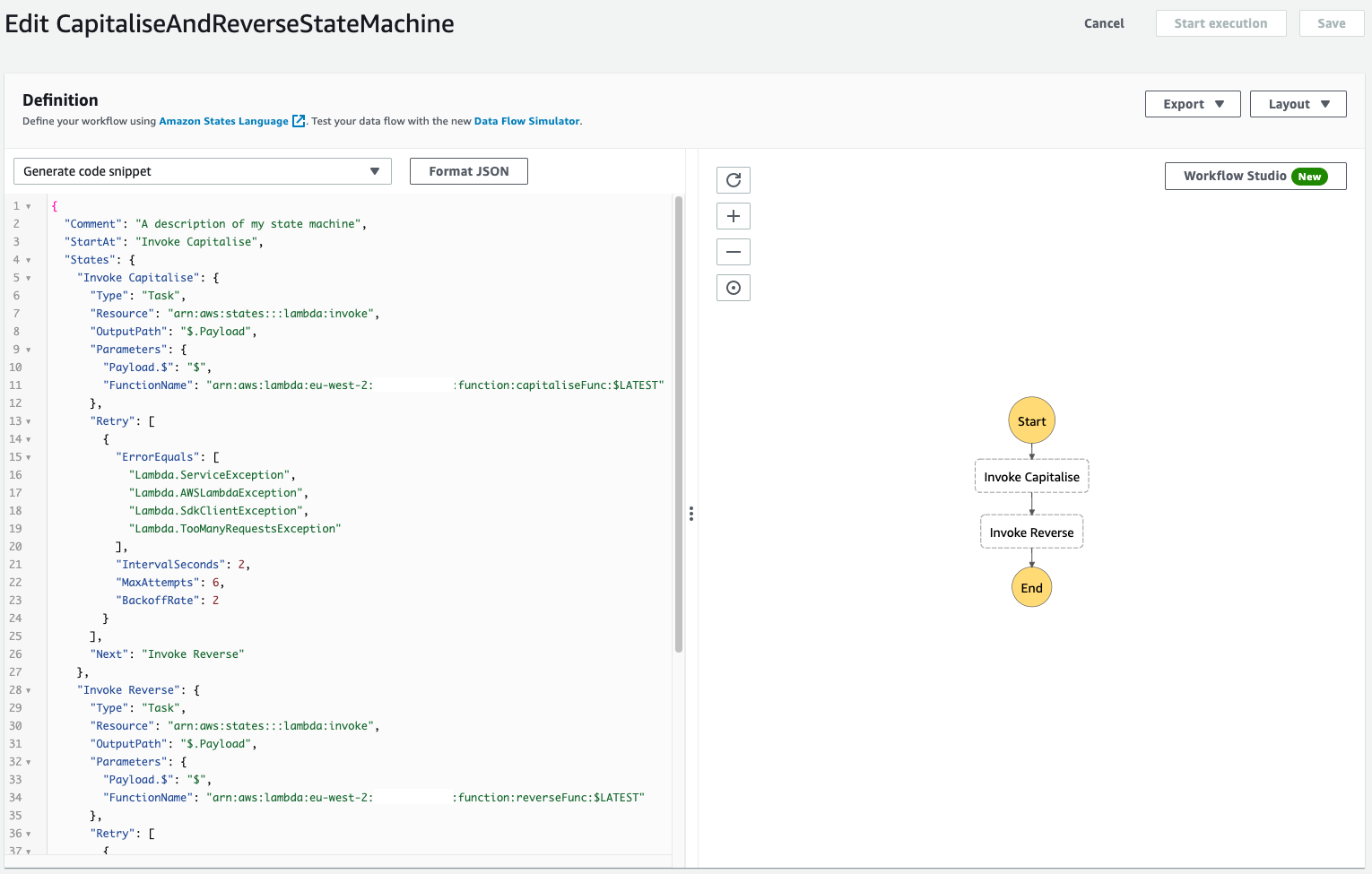


You can choose to design workflow visually for ease and use Express to make your functions work synchronously.



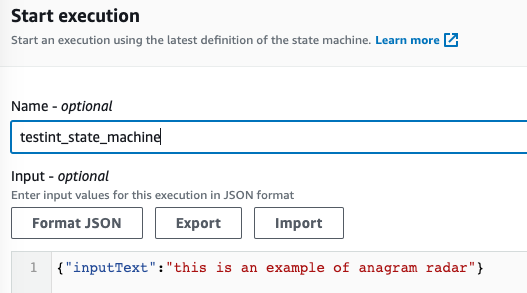


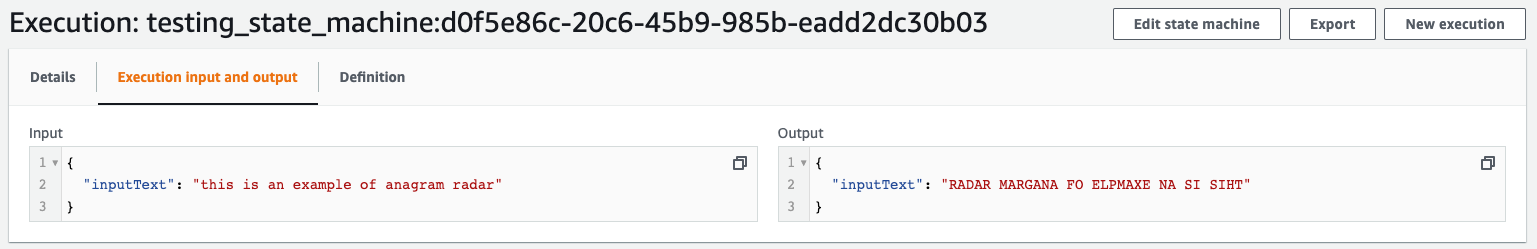


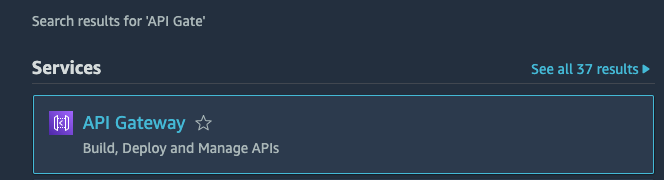


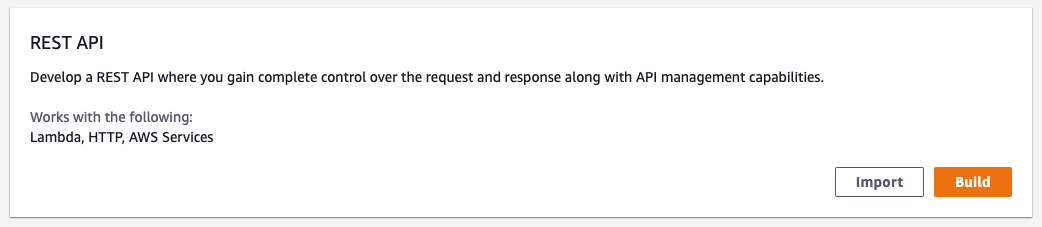
Click on New execution to test your State machine.

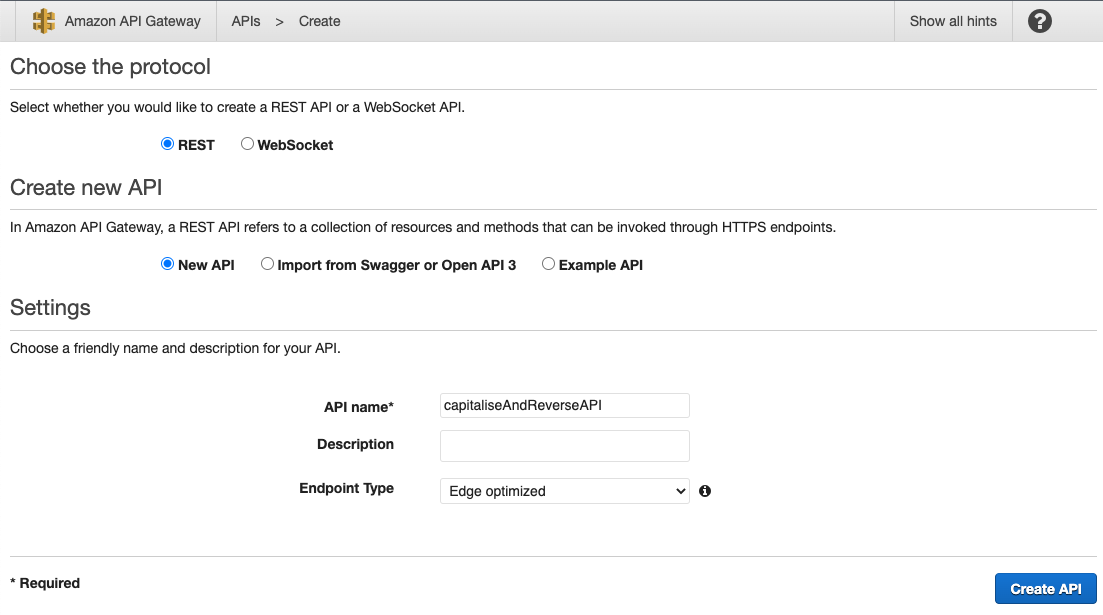


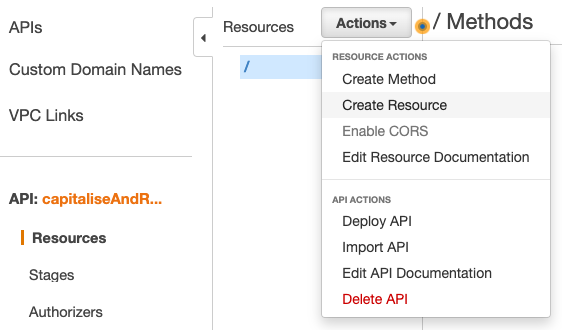


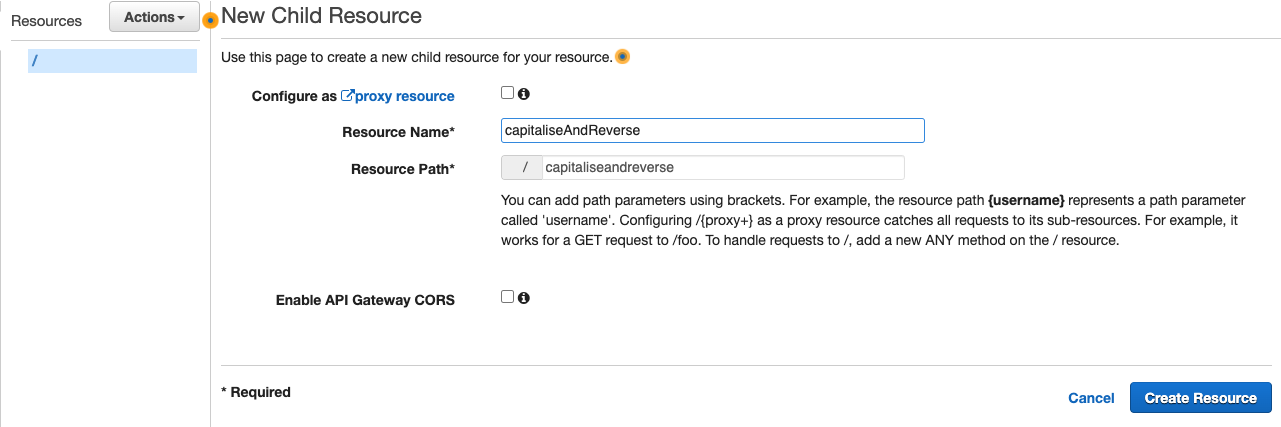


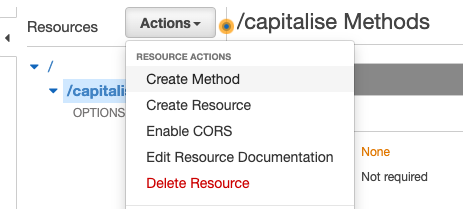




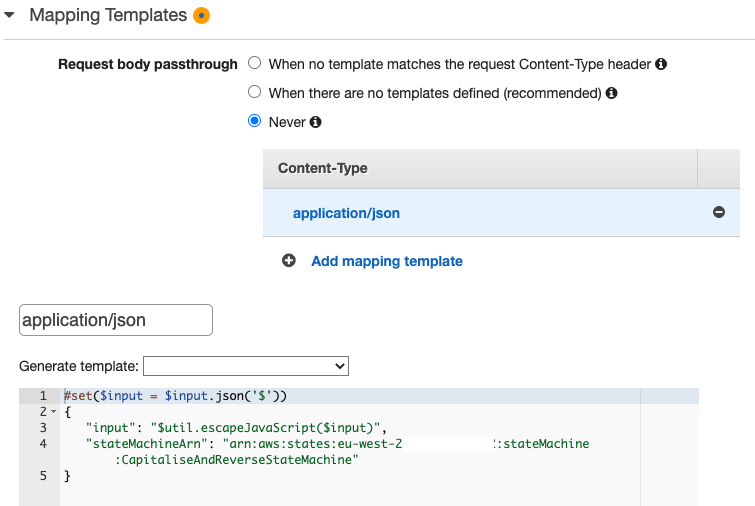




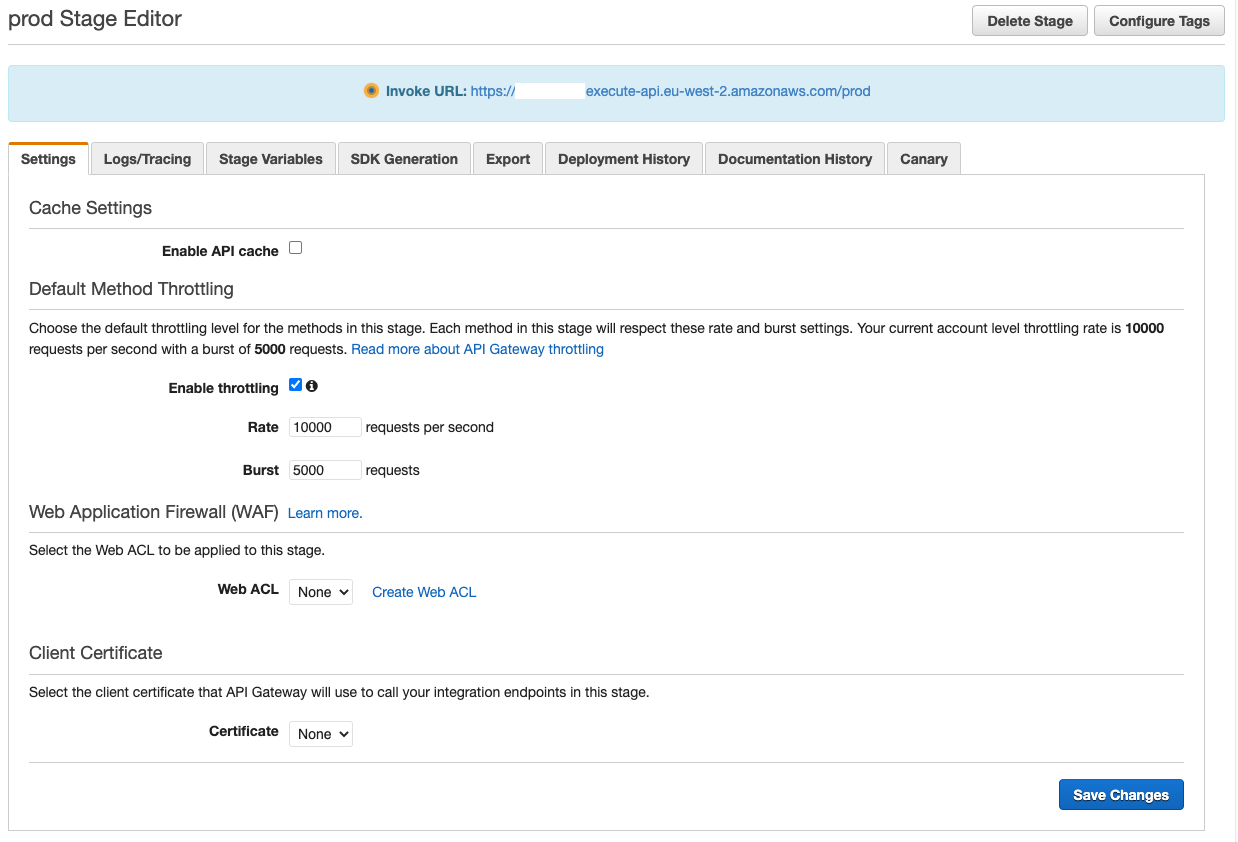








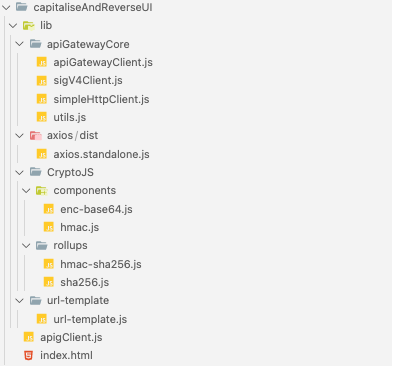
You then define the Stage. A Stage is a named reference to a deployment, which is a snapshot of the API. You use a Stage to manage and optimize a particular deployment. For example, you can configure Stage settings to enable caching, customize request throttling, configure logging, define stage variables, or attach a canary release for testing.



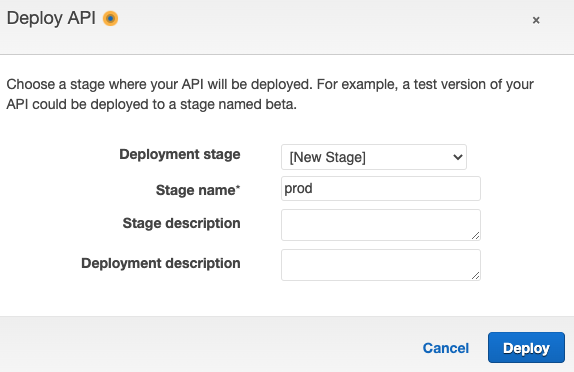
Generate the SDK, so you can use the generated code in your web app and call this API Gateway.



You then extract the generated JavaScript code as below:



And finally deploy the API (back in the AWS API Gateway section).



Your final HTML will looks like below; do notice that you have introduced a field to display your output.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Reverse and Capitalise with AWS Lambda</title>

<style>

body {

font-family: Verdana;

text-align: center;

}

form {

max-width: 500px;

margin: 50px auto;

padding: 30px 20px;

box-shadow: 2px 5px 10px rgba(0, 0, 0, 0.5);

}

.form-control {

text-align: left;

margin-bottom: 25px;

}

.form-control input {

padding: 10px;

display: block;

width: 95%;

}

</style>

</head>

<body>

<form id="form" onsubmit="callLambdaFunction(); return false;">

<div class="form-control">

<input type="text" id="inputText" placeholder="Enter some text here" />

</div>

<div class="form-control">

<button type="submit" value="submit">Submit</button>

</div>

<div class="form-control">

<input type="text" readonly id="outputText" placeholder="Output will appear here" />

</div>

</form>

<script type="text/javascript" src="lib/axios/dist/axios.standalone.js"></script>

<script type="text/javascript" src="lib/CryptoJS/rollups/hmac-sha256.js"></script>

<script type="text/javascript" src="lib/CryptoJS/rollups/sha256.js"></script>

<script type="text/javascript" src="lib/CryptoJS/components/hmac.js"></script>

<script type="text/javascript" src="lib/CryptoJS/components/enc-base64.js"></script>

<script type="text/javascript" src="lib/url-template/url-template.js"></script>

<script type="text/javascript" src="lib/apiGatewayCore/sigV4Client.js"></script>

<script type="text/javascript" src="lib/apiGatewayCore/apiGatewayClient.js"></script>

<script type="text/javascript" src="lib/apiGatewayCore/simpleHttpClient.js"></script>

<script type="text/javascript" src="lib/apiGatewayCore/utils.js"></script>

<script type="text/javascript" src="apigClient.js"></script>

<script type="text/javascript">

function callLambdaFunction() {

try {

var inputTextValue = document.getElementById("inputText").value;

var apigClient = apigClientFactory.newClient();

var params = {};

var body = {

inputText: inputTextValue,

};

apigClient

.capitaliseandreversePost(params, body)

.then(function (result) {

document.getElementById("outputText").value = JSON.parse(result.data.output).inputText;

})

.catch(function (result) {

console.log(result);

});

} catch (error) {

console.log(error);

}

return false;

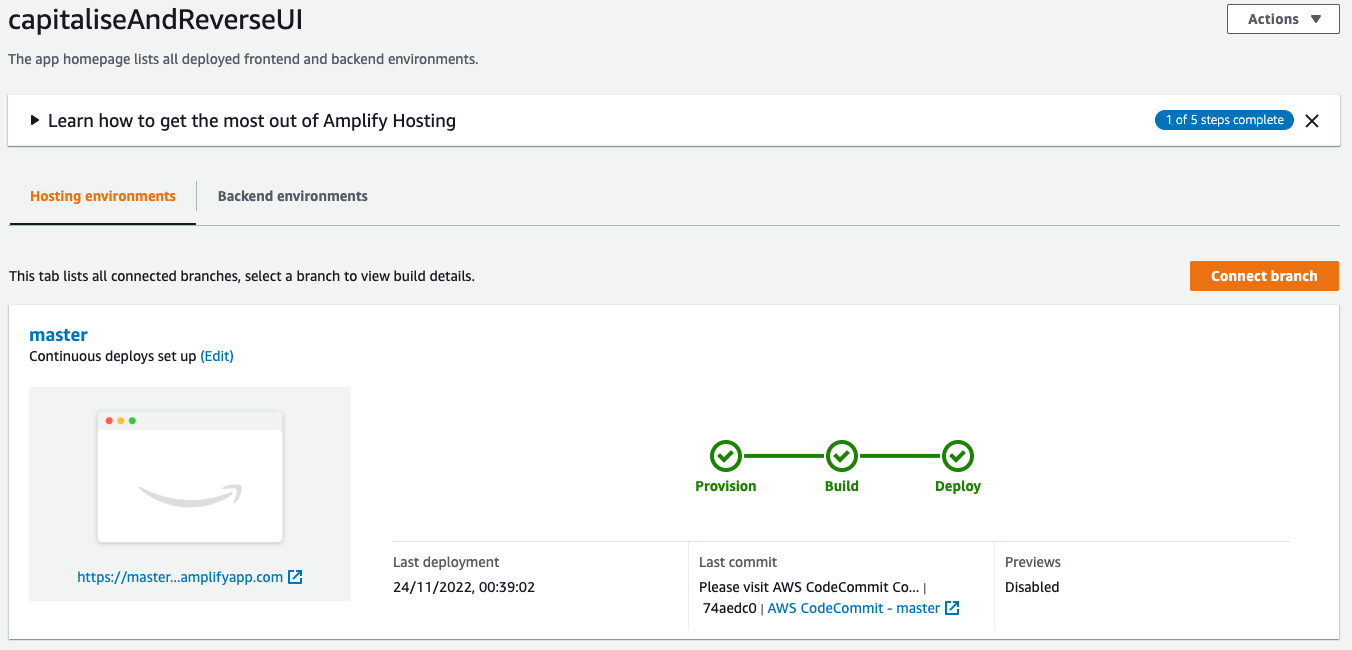
}

</script>

</body>

</html>

You then commit and push the changes to AWS CodeCommit repository and wait for it to be deployed.



And you can now test your web app by visiting the URL provided to you by AWS Amplify.

