**March 2020**



**Getting Started with   
Serverless Development on AWS**

*Hands-on with AWS CodeStar, AWS Cloud9 and AWS Lambda*

Table of Contents

[Overview 3](#_Toc502863953)

[Introduction 4](#_Toc502863954)

[Launching a development toolchain using AWS CodeStar 4](#_Toc502863955)

[Writing your first AWS Lambda Function on AWS Cloud9 7](#_Toc502863956)

[Deploying Your Function using AWS SAM and AWS CodeDeploy 12](#_Toc502863957)

[Debugging and Monitoring your function 14](#_Toc502863958)

[Clean up 17](#_Toc502863959)

[Conclusion 17](#_Toc502863960)

# Overview

With services like AWS CodeStar, AWS Cloud9, AWS Lambda and Amazon API Gateway, developers can very easily develop, debug, and deploy serverless applications in the cloud.

AWS CodeStar enables you to quickly develop, build, and deploy applications on AWS. AWS CodeStar provides a unified user interface, enabling you to easily manage your software development activities in one place. With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster.

AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. With Cloud9, you can quickly share your development environment with your team, allowing you to pair program and track each other's inputs in real-time.

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability.

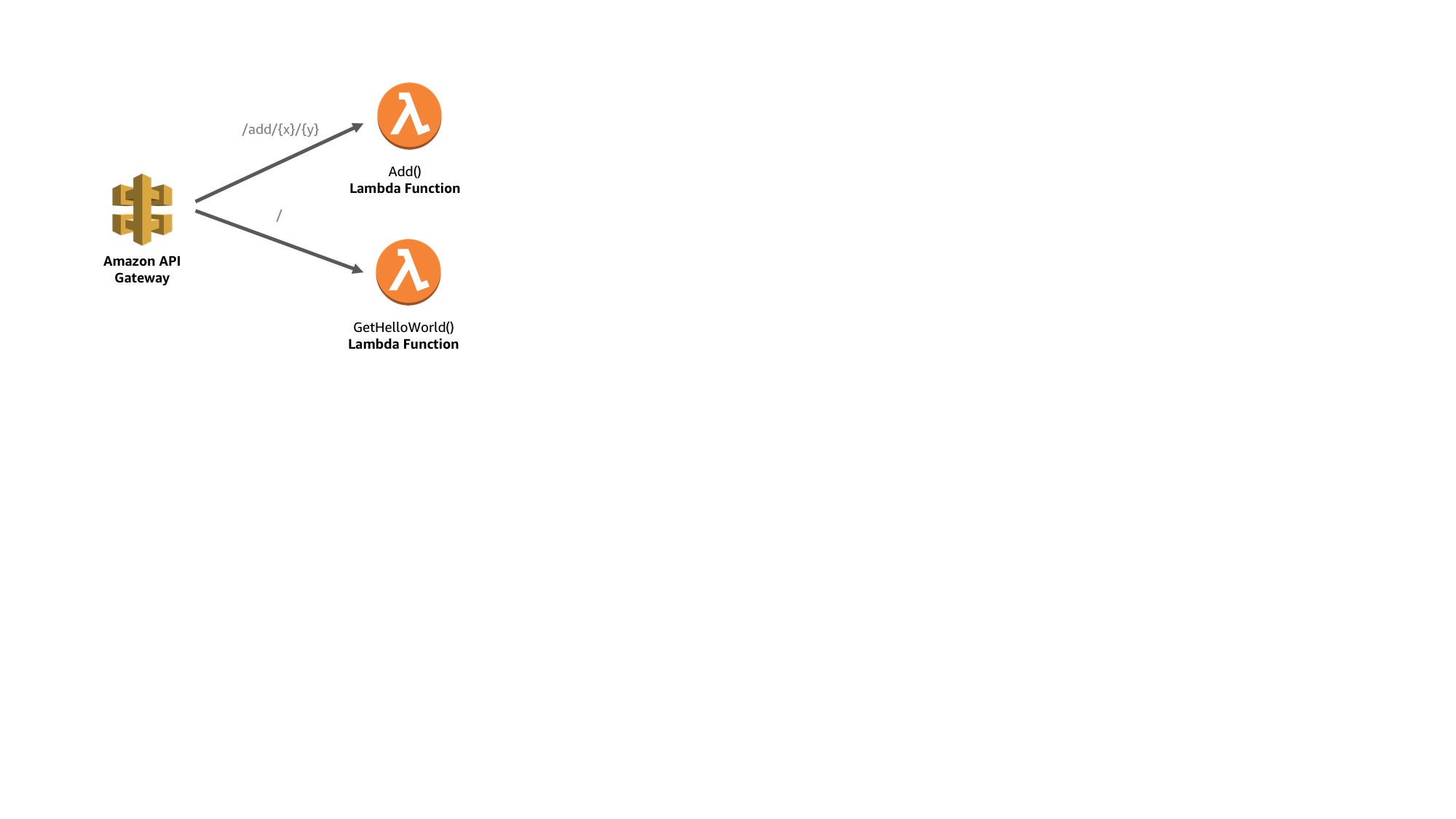
In this Lab, you will experience:

* Launching development tools for building a serverless application using AWS CodeStar.
* Developing your first AWS Lambda function.
* Deploying your first AWS Lambda function using AWS SAM (Serverless Application Model).
* Testing and debugging your AWS Lambda function in AWS Cloud9 and AWS SAM Local.

**Important:** Many of the steps in this lab involve resolving errors that have been built in to the provided starter files (in step 17). This approach will show how the IDE can be used to develop, test and deploy serverless applications.

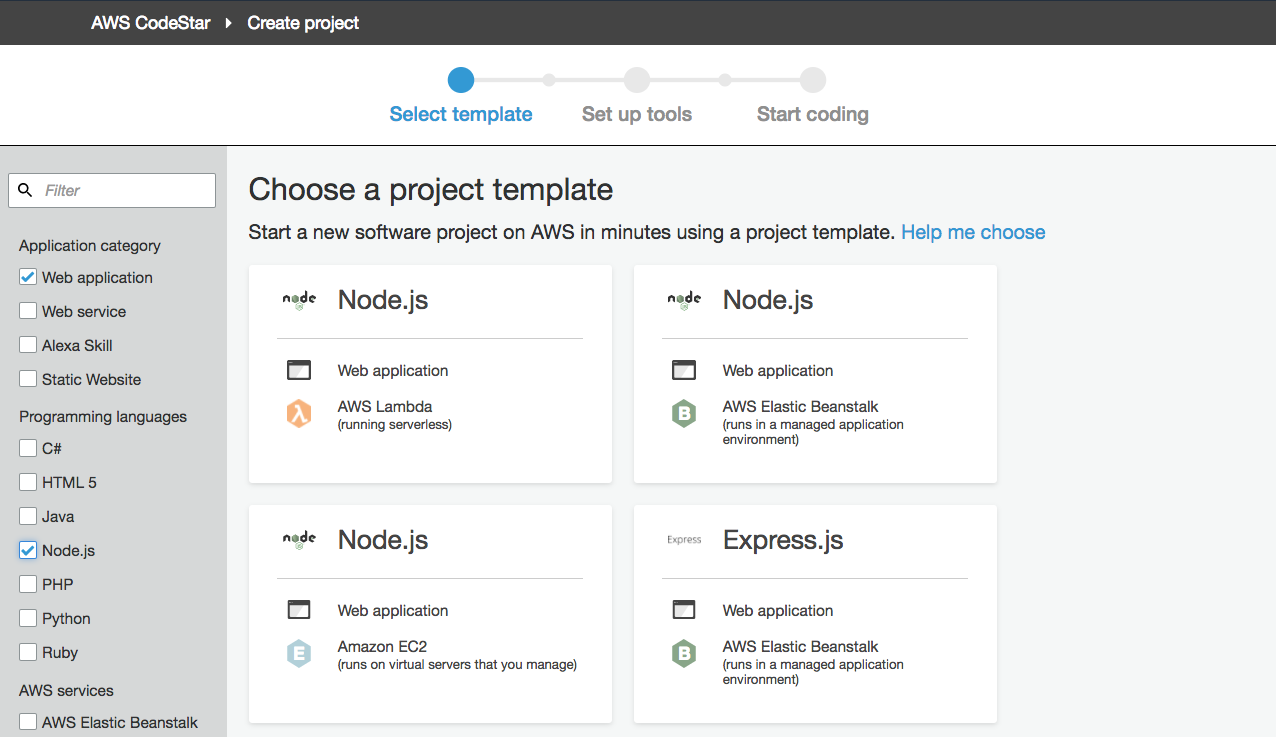
# Introduction

In this hands-on lab, we are going to start with a Hello World node.js serverless app that returns a static web page. As part of this lab, we will be developing, testing, debugging and deploying a new serverless function for adding 2 numbers. The high level architecture is as follows:

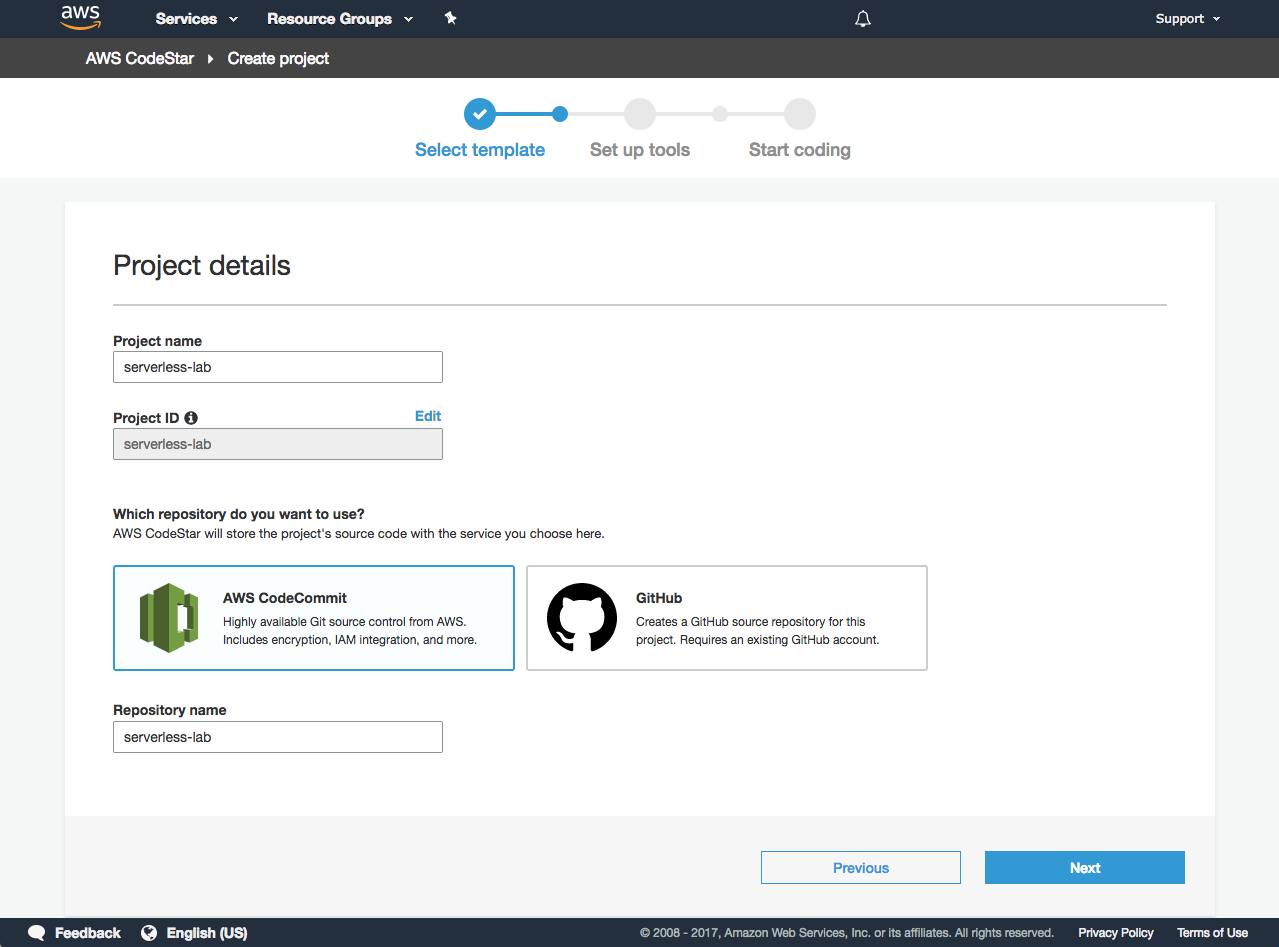


# Launching a development toolchain using AWS CodeStar

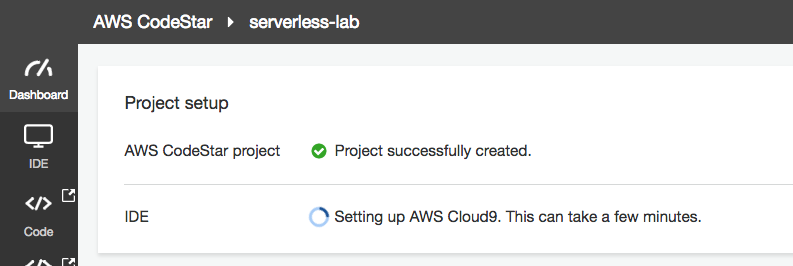
1. Sign into the AWS Management Console <https://console.aws.amazon.com/>.   
     
   **Note:** For this lab you will need to use an IAM user and not a federated user account. The account will require *AWSCodeStarFullAccess* managed policy. Alternatively, you can create projects if you have an IAM administrative user with full permissions for all AWS Services (*AdministratorAccess* managed policy). For more information, see [Setting up AWS Code Star (Step 3: Create or Use an IAM User).](https://docs.aws.amazon.com/codestar/latest/userguide/setting-up.html#setting-up-create-iam-user)
2. In the upper-right corner of the AWS Management Console, confirm you are in the desired AWS region (e.g., N. Virginia).
3. Click on **AWS CodeStar** from the list of all services.
4. Click on **Start a project**.
5. If this is the first time you use the service, you will be prompted to create the required service roles for AWS Code \* services. Select **Yes, create role**.
6. Choose a project template:Under *Application Category* select *Web Application.*Under *Programming Languages* select *Node.js.*  
   Pick *Node.JS Web Application AWS Lambda (running serverless).*



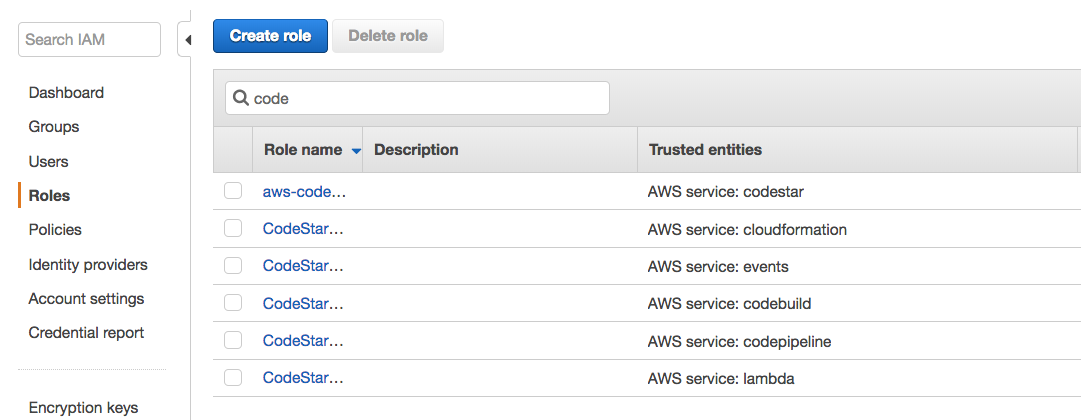
1. Enter the project details:  
   Project name: *serverless-lab.*Which repository do you want to use? *AWS CodeCommit.*  
   Click **Next**.



1. Leave everything as the default and click on **Create Project**. If this is the first time you use the service, you will also be prompted to enter your display name and email.
2. We are going to use *AWS Cloud9* as our IDE. Select AWS Cloud9 and hit **Next.**
3. For our instance, we will select *t2.small*. We will leave the networking settings as default which will launch the instance in our default VPC in a public subnet. Under *Cost-saving* settings, observe that the environment will be automatically shut down after 30 minutes. Click **Next**.
4. AWS CodeStar is now provisioning all the AWS Code \* services. This process may take around 3-5 minutes.



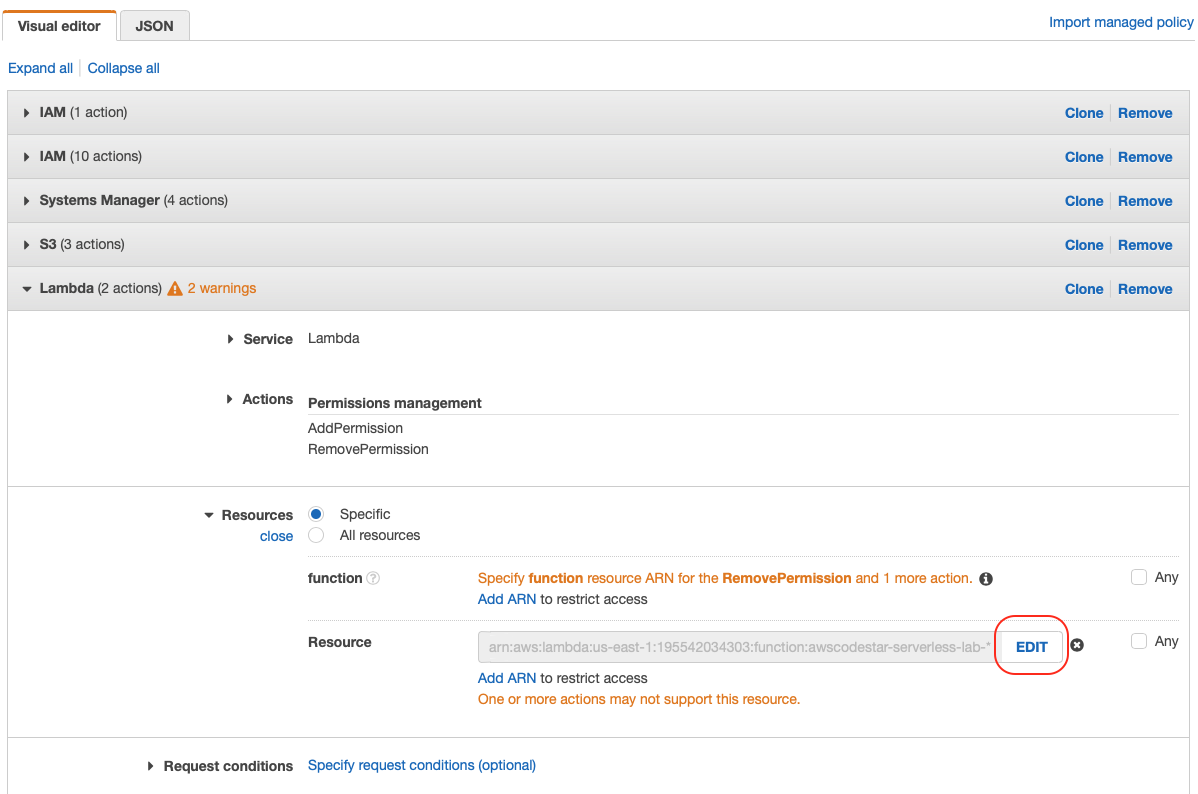
1. While you wait, open up a new browser tab and go to the [IAM Roles console](https://console.aws.amazon.com/iam/home?region=us-east-1#/roles) (Services -> Security -> IAM) (listed under Services). Search for *‘code’* and notice that AWS CodeStar created new IAM Roles for each of the AWS service we are going to use.

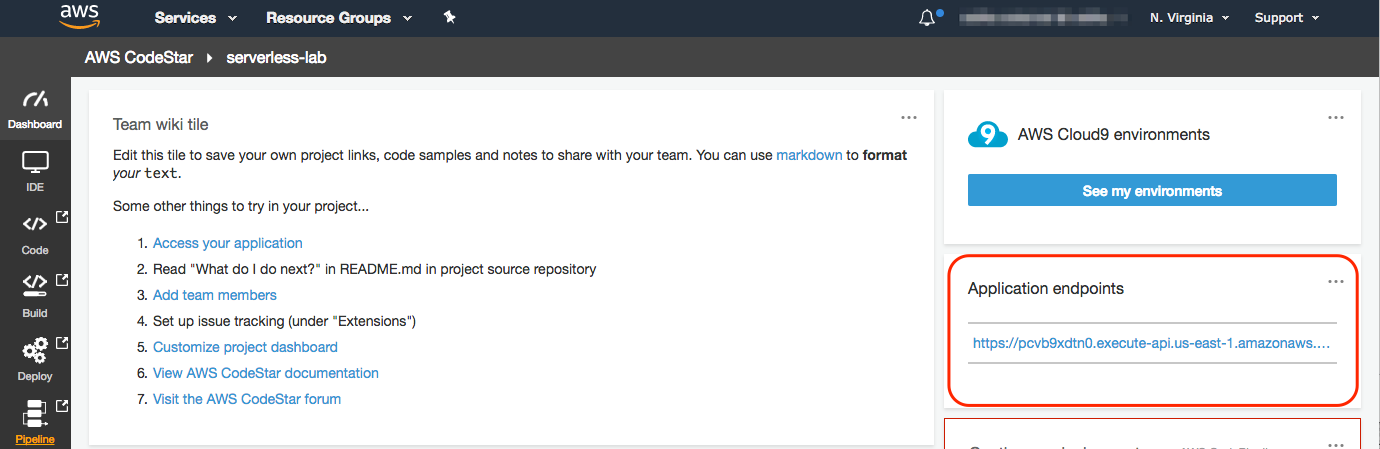


Click on CodeStarWorker-serverless-lab-CloudFormation role, expand Permission policies and click on Edit policy

Click on Edit button for Resource and update resource with following value

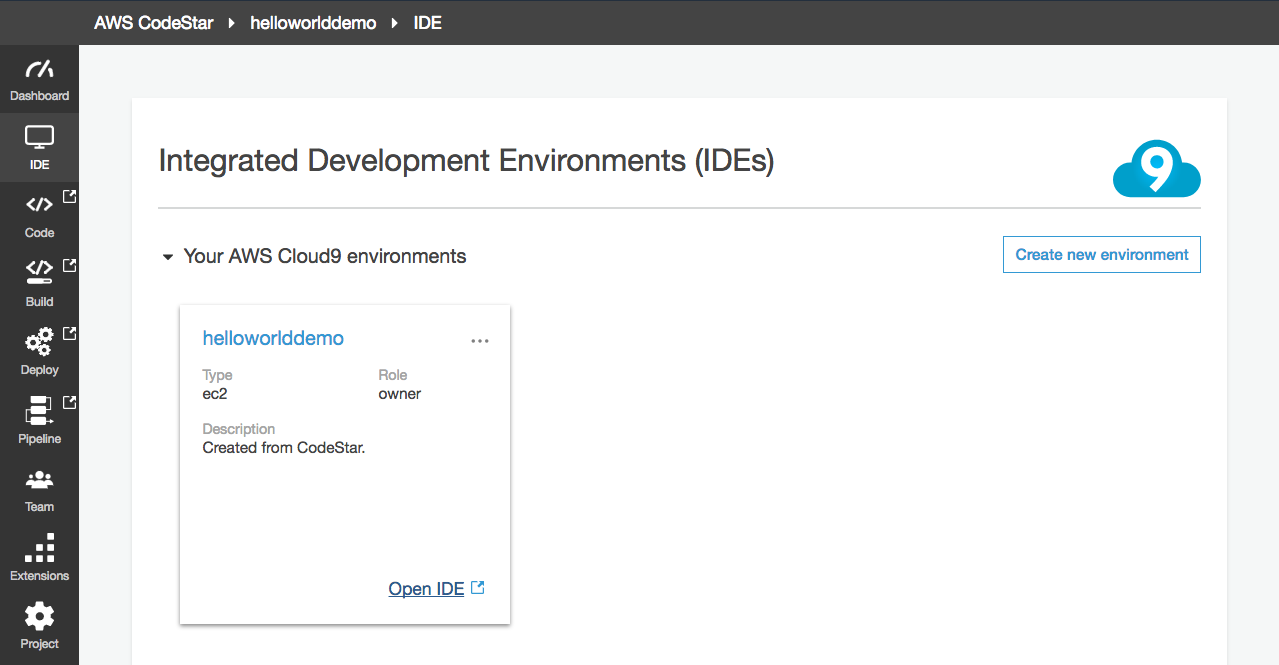
arn:aws:lambda:us-east-1:”your\_account\_number”:function:awscodestar-serverless-lab-\*



1. Head back to the *AWS CodeStar Dashboard* and scroll to the *Application endpoints* panel.   
     
   
2. Click on the endpoint URL. You should see a “Hello World” web page rendered by Node.js. Congratulations! You successfully configured an end-to-end development and continuous deployment pipeline on AWS.

# Writing your first AWS Lambda Function on AWS Cloud9

1. Go back to the AWS CodeStar dashboard, and click on **IDE** on the left pane.   
   Click **Open IDE.**



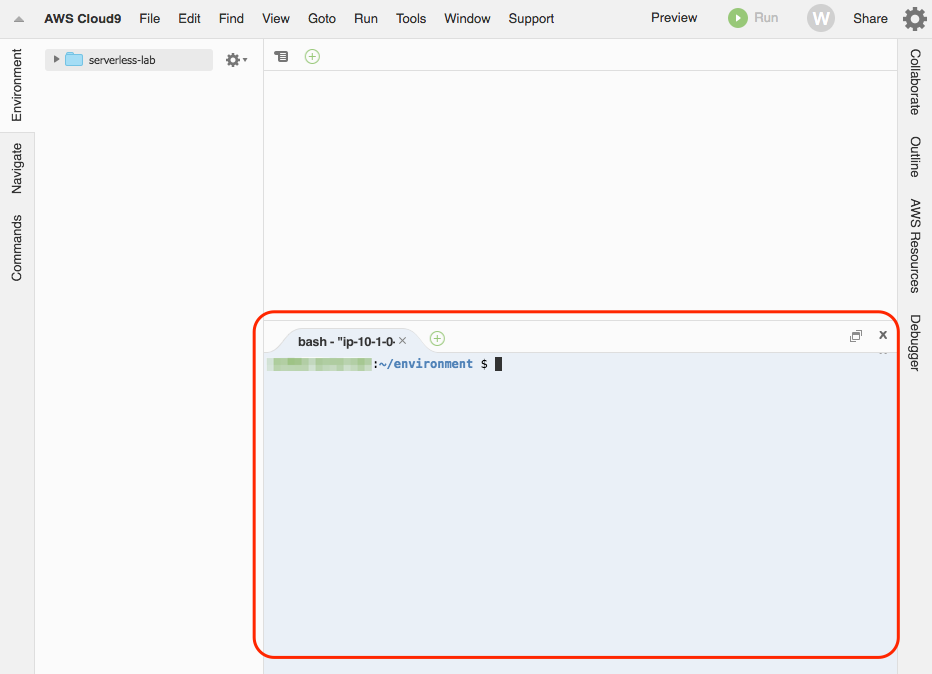
|  |  |
| --- | --- |
| **Note:** | You use the AWS Cloud9 IDE, running in a web browser on your local computer, to interact with your environment. An Amazon EC2 instance or your own server connects to the environment. An environment is a place where you store your project's files and where you run the tools to develop your apps.  You use the AWS Cloud9 IDE to work with files in the environment. You can:   * Store these files locally on the instance or server. * Clone a remote code repository—such as a repo in AWS CodeCommit—into your environment. * Work with a combination of local and cloned files in the environment. |

1. Upon first login, AWS Cloud9 automatically clone a starter project “locally” into our development instance. You should see something like this:

|  |
| --- |
| user:~/environment $ /tmp/git-cloning-runner-xxx-xxx.sh  Cloning into '/home/ec2-user/environment/serverless-lab'...  remote: Counting objects: 16, done.  Unpacking objects: 100% (16/16), done.  Navigate to your cloned repository by typing "cd /home/ec2-user/environment/serverless-lab" to start working with "https://git-codecommit.us-east-1.amazonaws.com/v1/repos/serverless-lab"  To set your display name run "git config --global user.name YOUR\_USER\_NAME"  To set your display email run "git config --global user.email YOUR\_EMAIL\_ADDRESS"  user:~/environment $ |

1. We’re going to create a new API microservice in this project. Perform the following command in the terminal window at the bottom (labeled as *bash – “ip-xx-xx-xx-xx”):*  
     
   cd serverless-lab  
   wget <https://s3-us-west-2.amazonaws.com/apn-bootcamps/serverless-2018/addservice-01.tar.gz>  
   tar xf addservice-01.tar.gz  
   rm addservice-01.tar.gz

**Note:** In some browsers (E.g. Google Chrome), you may be prompted to download Cloud9 Browser Extension to enable copy-paste between this document, and the AWS Cloud9 IDE inside the browser window.



The commands above will add the following files to your local AWS Cloud9 environment:

* 1. add.js – our addition-as-a-service lambda function
  2. test/event.numbers.json – the test payload
  3. .gitignore – instruction for git to ignore temp files

To confirm we have all the files in place, perform ls in the terminal window and you should see the following files:

|  |  |
| --- | --- |
| * add.js * buildspec.yml * index.js * \nothing | * \public * README.md * template.yml * \test |

1. We will need to update our SAM template (*template.yml*) to register our new function.   
   Incorporate these \*\***requirements\*\*** by modifying the template.yml file:
2. Function name: “AddService”
3. The functionality will be handled by the handler() function in add.js
4. Runtime: “nodejs12.x”
5. Use the same LambdaExecutionRole
6. The event trigger will be coming from our existing “GetEvent” API Gateway
7. We would like to return a JSON response whenever we receive an *ANY* HTTP request to the API with path /add/{x}/{y}. Therefore, given <http://[api-address]/add/1/2>,the API should return { … “result”:3 … }.

**Hint:** Use the *GetHelloWorld* resource definition in the same file (template.yml) to help you apply the necessary changes. See [Serverless Resources Within AWS SAM](http://docs.aws.amazon.com/lambda/latest/dg/serverless_app.html) for the complete reference, or use the guide below:

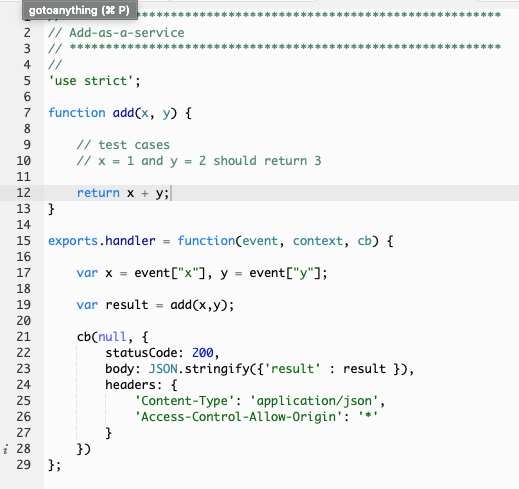
|  |
| --- |
| AddService:  Type: AWS::Serverless::Function  Properties:  Handler: add.handler  Runtime: nodejs12.x  Role:  Fn::GetAtt:  - LambdaExecutionRole  - Arn  Events:  GetEvent:  Type: Api  Properties:  Path: /add/{x}/{y}  Method: any |

1. After you are done making those changes. Let’s test our resource definition locally in our Cloud9 environment to make sure we have defined our new AWS Lambda function correctly. In the terminal window, confirm that you are in  
   /home/ec2-user/environment/serverless-lab/ and perform the following:  
     
   sam local invoke "AddService" -e test/event.numbers.json

Our add.js currently do not implement the functionality we are seeking. We are expecting to see “Error: Not Implemented” exception. This means we have configured our SAM template correctly.

**Note:** The first time you execute a function, SAM LOCAL will fetch the appropriate Docker image for the function as defined in our template.yml. Once that is done, SAM LOCAL will invoke the Lambda function we just wrote, showed the response, *Duration* and the *Max Memory Used* metric which is useful for tuning our AWS Lambda function configuration.

1. Open up *add.js* and implement our add function in line 12-16.



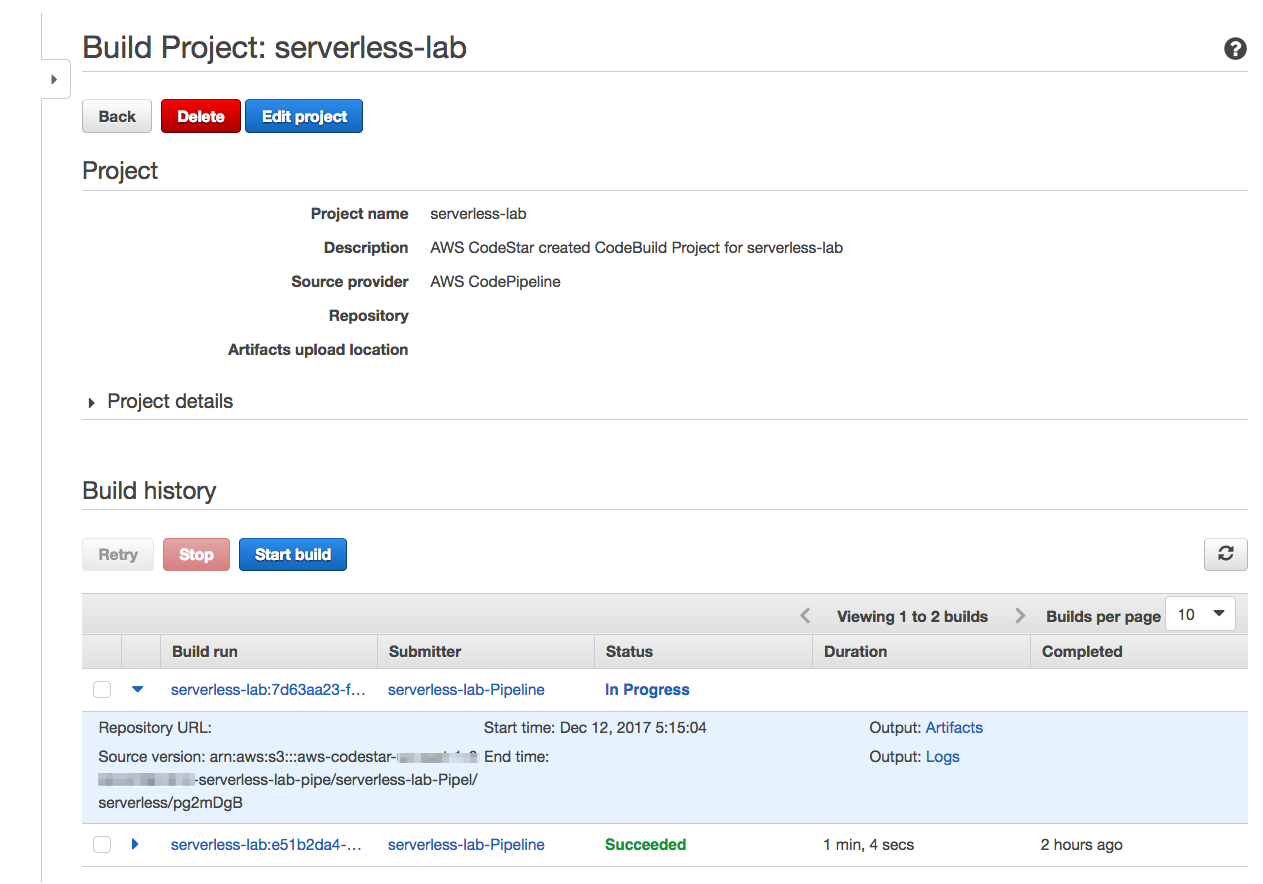
1. You think you got it right? Let’s test out the Lambda function locally again:  
     
   sam local invoke "AddService" -e test/event.numbers.json   
     
   should return the following:

{  
 "statusCode":200,  
 **"body":"{\"result\":116}",**  
 "headers":{"Content-Type":"application/json, … }  
}

1. Congratulations! You have implemented add()as an AWS Lambda function.

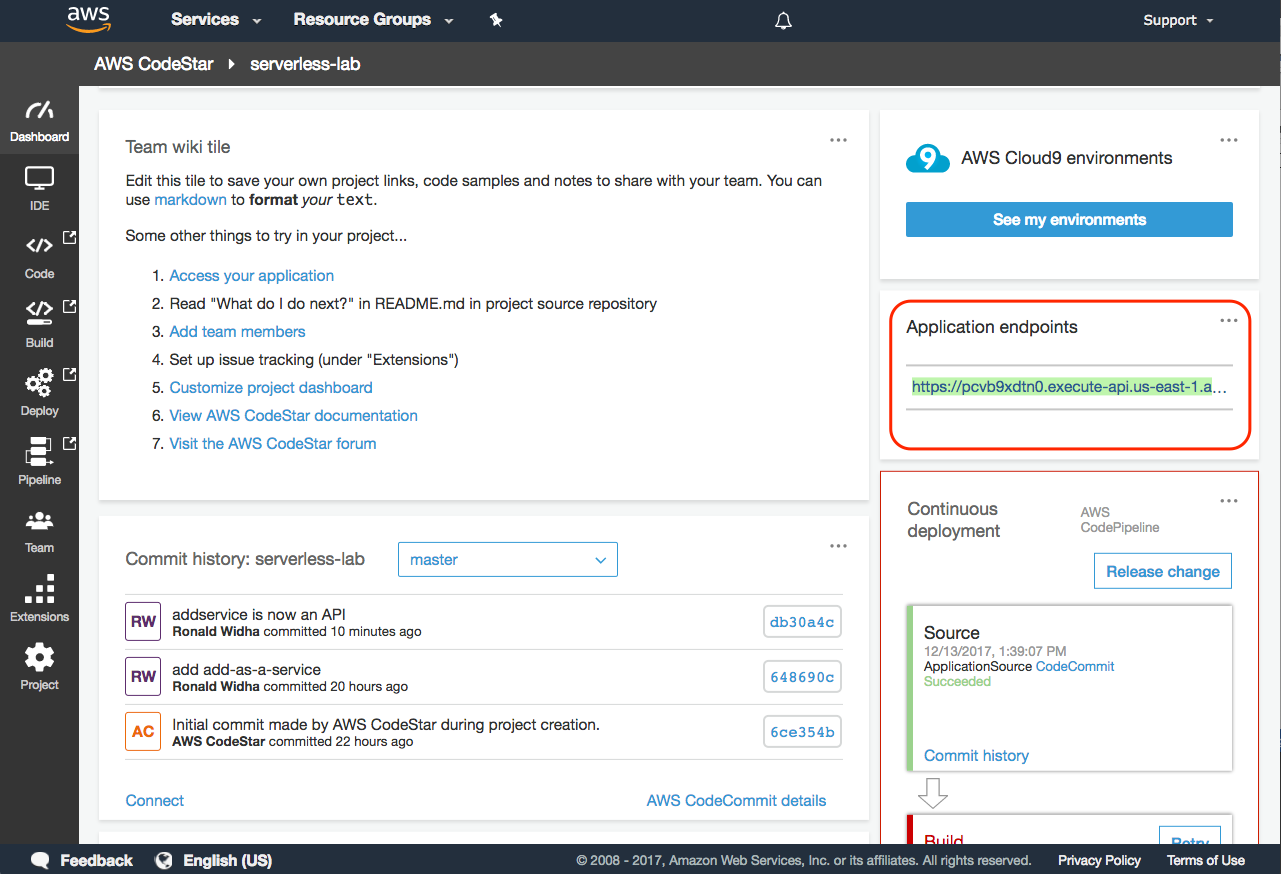
# Deploying Your Function using AWS SAM and AWS CodeDeploy

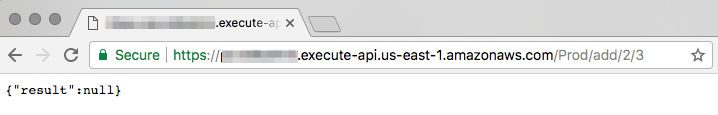
1. We will now configure our git user in the AWS Cloud9 environment so we can commit our changes to the code repository  
     
   git config --global user.email [you@example.com](mailto:you@example.com)  
   git config --global user.name "Your Name"
2. We are going to add all our pending changes, commit it to Git and push it to our AWS CodeCommit Repository:  
     
   git add -A  
   git commit -m "add add-as-a-service"  
   git push origin master
3. By default, AWS CodeStar configured AWS CodeBuild to deploy on every code commits. Go back to your AWS CodeStar serverless-lab dashboard. You can see the status of the current build, or see more details about the build by selecting **Build** from the left hand pane. You will notice that a build is currently taking place. This will take around 2 minutes.

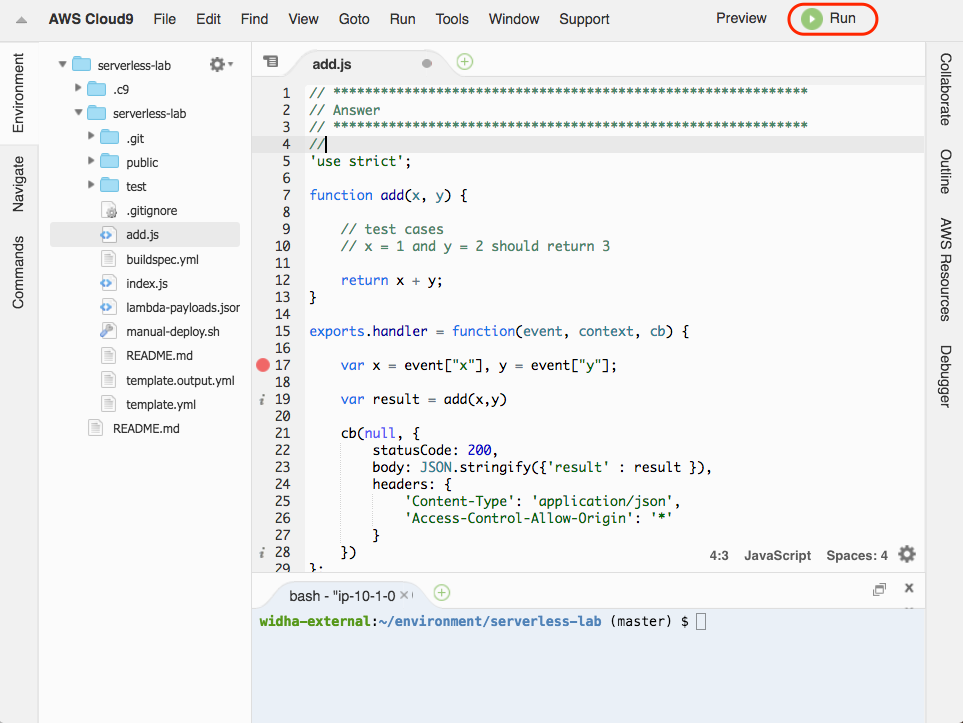


# Debugging and Monitoring your function

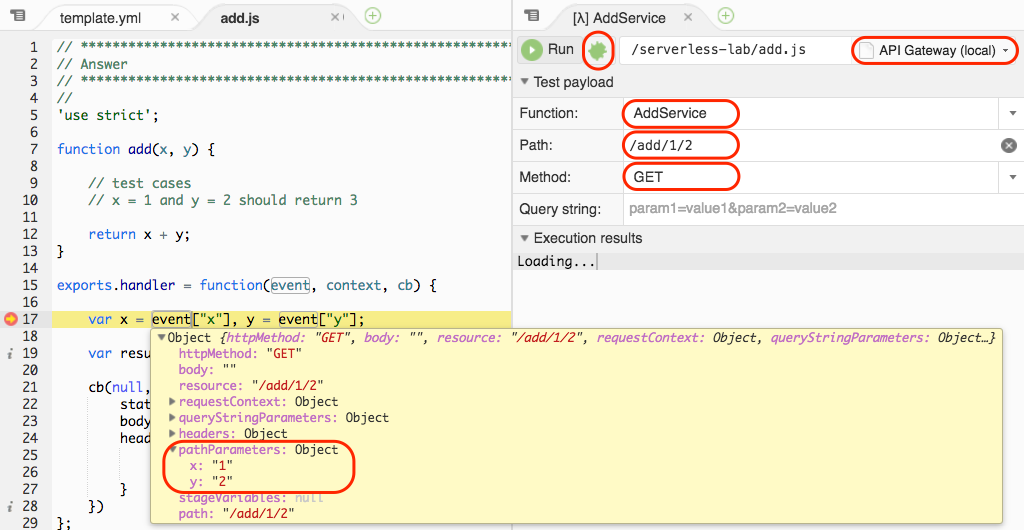
1. After a successful deployment, open the AWS CodeStar project dashboard and copy our *Application endpoints* location.



1. Open up a browser and modify the URL to look something like this:  
   <https://xxxx.execute-api.us-east-1.amazonaws.com/Prod/add/2/3>   
     
   Oh no! It looks like we have a bug. On step 19, we tested the Lambda function add.js in isolation and it performed correctly. However we have not tested the function integrated with the Amazon API Gateway. Head back to our AWS Cloud9 IDE so we can debug this issue.
2. Open add.js and add a breakpoint on line 17 by clicking on the space next to the row number until it is showing a red dot. Click **Run** to start our debugging session.



1. Since the bug only appears when the function is called by Amazon API Gateway, we will debug our function using a local API Gateway environment. Enter the following configuration:
   1. Click on the Debug icon
   2. Run on: *API Gateway (local)*
   3. Function: *AddService*
   4. Path: */add/1/2*
   5. Method: *GET*
   6. Click **Run**



1. We should now see the debugger stops at line 17. Observing it closely, we need to make adjustment to line 17 to read from the correct event object properties. Make those changes.  
     
   **Hint:** Watch out for the [data types](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number#Examples).



1. Let’s test it again using the command line this time:  
     
   sam local start-api&

Press enter.  
  
curl http://127.0.0.1:3000/add/25/75

We should now receive the right result { … "result":100 … }

1. Commit our latest changes to AWS CodeCommit again.  
   git add -A  
   git commit -m “Bugfix: Add() now performs correctly on API GW”  
   git push origin master
2. Wait for our Continuous Deployment pipeline to complete, and test it from our live endpoint that looks similar to: <https://xxxx.execute-api.us-east-1.amazonaws.com/Prod/add/25/75>
3. Congratulations, you have completed the lab.

# Clean up

After you are done with your lab, head over to the CloudFormation console (listed under Services) and delete aws-cloud9-serverless-lab-xxxxxx, awscodestar-serverless-lab-lambda, and awscodestar-server-lab stacks. This will delete all the resources we created during this lab.

# Conclusion

In this lab you have learned creating end-to-end development tools using AWS CloudStar, writing your first Serverless microservice using AWS Lambda and Amazon API Gateway using the AWS Cloud9 IDE