GETTING STARTED WITH SERVERLESS

by *P.Lokesh(Mech)*

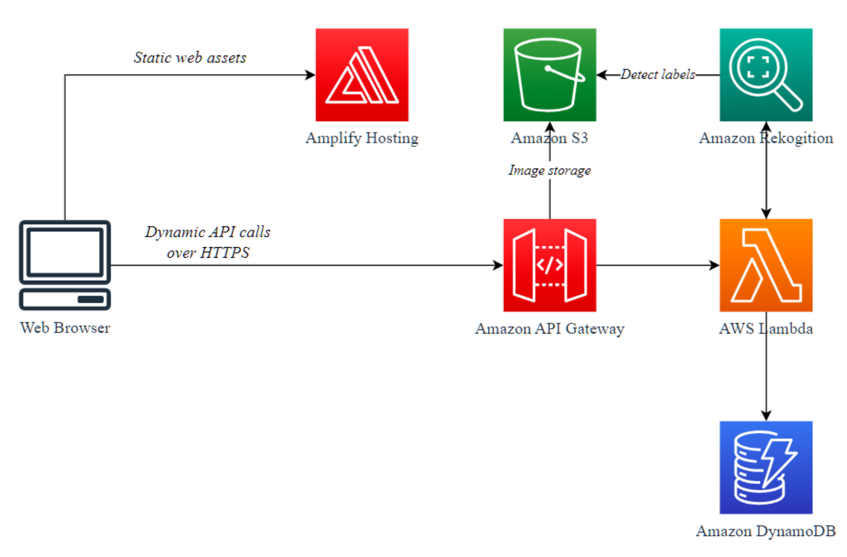
Overview

In this tutorial, you'll create a simple serverless web application that implements a "Todo app" with an API to store and retrieve tasks in a cloud database. In addition, we will integrate machine learning to automatically identify and label objects in images attached to tasks.

- AWS Experience: Beginner (Basic HTML/CSS/JavaScript and Linux command line knowledge is helpful)
- Time to Complete: 2 hours
- Tutorial Prerequisites: An AWS account, a text editor, recommended browser: latest version of Chrome or Firefox, Amazon S3 , AWS Lambda , Amazon API Gateway , AWS Amplify , Amazon DynamoDB , Amazon Rekognition , AWS Cloud9 .

- ## Application Architecture

- The application architecture uses AWS Lambda, Amazon API Gateway, Amazon DynamoDB, Amazon Simple Storage Service (S3), and AWS Amplify Console. Amplify Console provides continuous deployment and hosting of the static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser. JavaScript executed in the browser sends and receives data from a public backend API built using Lambda and API Gateway. DynamoDB provides a persistence layer where data can be stored by the API's Lambda function. S3 is used to store uploaded images. Finally, Amazon Rekognition is used to detect and label objects in those images.



Modules:

This workshop is divided into multiple steps. Each step describes a scenario of what we're going to build and step-by-step directions to help you implement the architecture and verify your work. The steps must be performed in order for the workshop to be successful.

1. Install prerequisites
2. Build a serverless backend
3. Configure a Lambda Authorizer
4. Build and deploy the web application
5. Test the application
6. Configure Amazon Rekognition Integration
7. Terminate Resources.

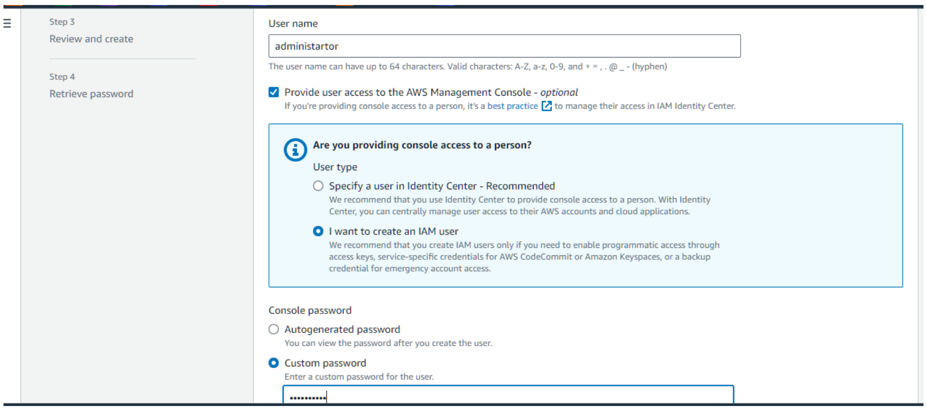
Before starting the workshop...

Preparing for the workshop

- **Start with an AWS Account**
- When you are ready with the account preparation, you may start the lab from the next page.
- Also, to minimize the browser compatibility issue, we recommend you to use either Mozilla Firefox or Google Chrome browser for the lab.
- **Create an AWS account**

If you already have an AWS account or create an AWS account, create an IAM user that has access to AWS account. Log in to the AWS account, you can create a IAM user using IAM console. Create an IAM user with Administrator role as below the step. If you already have an IAM user with administrator role, skip the next task.

1. From the Log in Page, log in with AWS root account email address and password then go to IAM Console
2. From the left sidebar of the IAM Console, click Users and then click Add user
3. Enter User name as Administrator.
4. Tick AWS Management Console access check box from Access type and then choose Custom password and enter the password
5. Click Next: Permissions



User name

administartor

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

[✓] Provide user access to the AWS Management Console - *optional*

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

Are you providing console access to a person?

User type

○ Specify a user in Identity Center - Recommended

We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

[✓ I want to create an IAM user]

We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

○ Autogenerated password

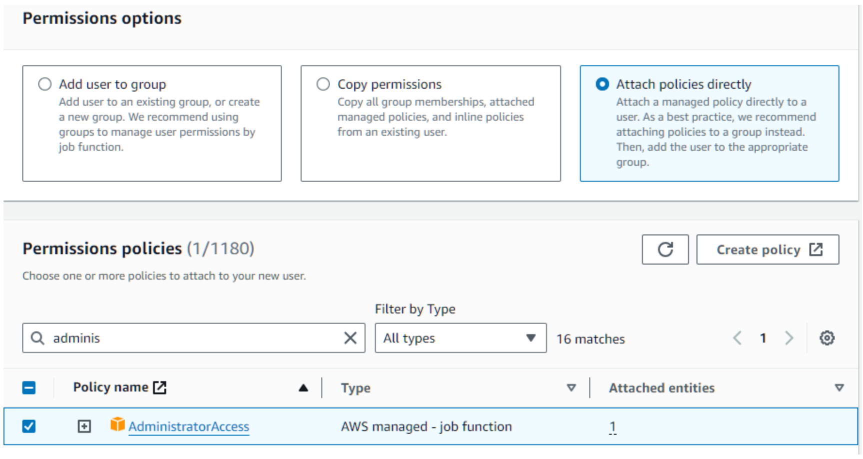
You can view the password after you create the user.

[✓ Custom password]

Enter a custom password for the user.

••••••••••

6. Choose Attach existing policies directly, tick AdministratorAccess option and then click **Next: Tags**.



Permissions options

- Add user to group: Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions: Copy all group memberships, attached managed policies, and inline policies from an existing user.
- [✓ Attach policies directly]: Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1180)

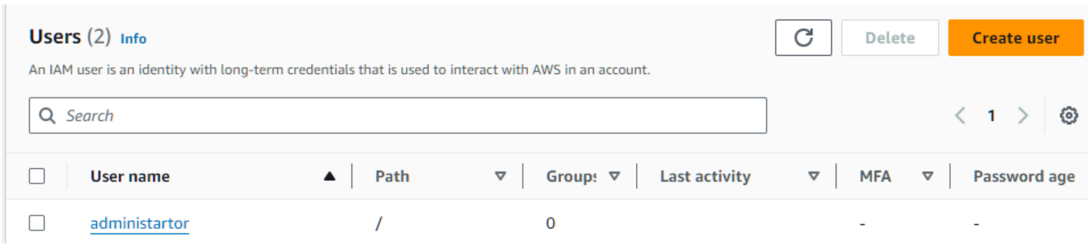
Choose one or more policies to attach to your new user.

Filter by Type: All types — search: adminis — 16 matches

	Policy name	Type	Attached entities
✓	AdministratorAccess	AWS managed - job function	1

7. Click Next: Review from Add tags(optional) section.
8. Review AdministratorAccess, AWS managed policy, is added to Administrator user and then click Create user
9. After creating the user, copy the login URL. The format of the URL is as below.

   https://975049895916.signin.aws.amazon.com/console



Users (2) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

	User name	Path	Groups	Last activity	MFA	Password age
☐	administartor	/	0		-	-

10. Log out from the root user and then log in to the Administrator user you just created using the URL you copied above.

Additional settings

Prerequisites

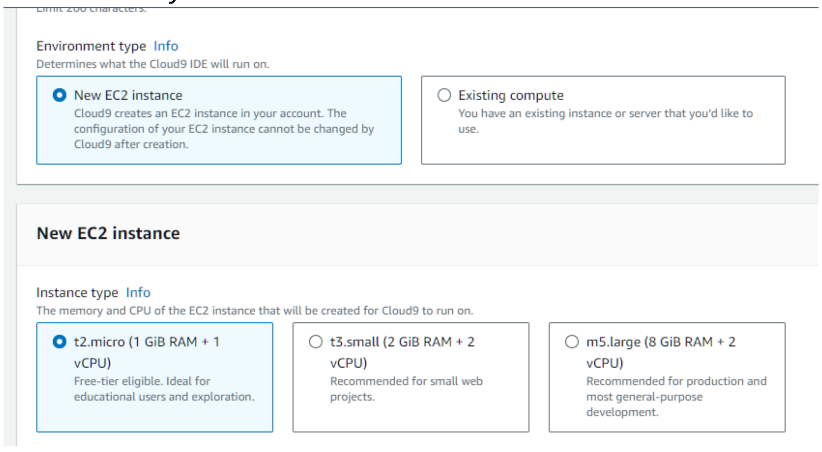
The following items are preliminary information for this workshop.

1. The images attached to this page were created to help carry out your hands-on lab. The **Identifier (ID)** of each resource you create (IAM role, S3 bucket, etc.) during this lab is different for each user account.
2. For AWS services' faster response, we recommend you to select **the nearest Region** during this workshop.

Create a Cloud9 Workspace

Launch Cloud9

1. Navigate to the Cloud9 console
2. Ensure you are using the appropriate region for the workshop, and switch the region if required.
3. Select **Create environment**
4. Name it **serverless-workshop**, and select **Next Step**
5. Ensure that you select **t2.micro** for **Instance size**.



Environment type Info
Determines what the Cloud9 IDE will run on.

New EC2 instance [✓]
Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

Existing compute
You have an existing instance or server that you'd like to use.

New EC2 instance

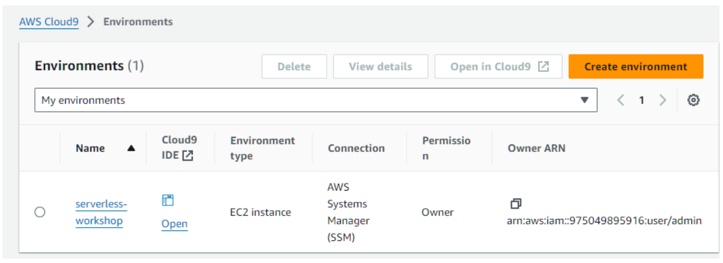
Instance type Info
The memory and CPU of the EC2 instance that will be created for Cloud9 to run on.

t2.micro (1 GiB RAM + 1 vCPU) [✓]
Free-tier eligible. Ideal for educational users and exploration.

t3.small (2 GiB RAM + 2 vCPU)
Recommended for small web projects.

m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and most general-purpose development.

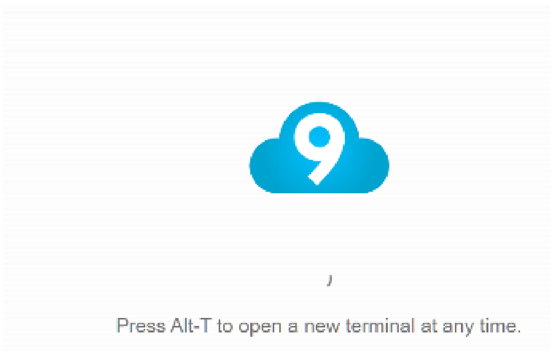
6. Keep the rest of the default settings, and select **Next Step**
7. Lastly, select **Create Environment**



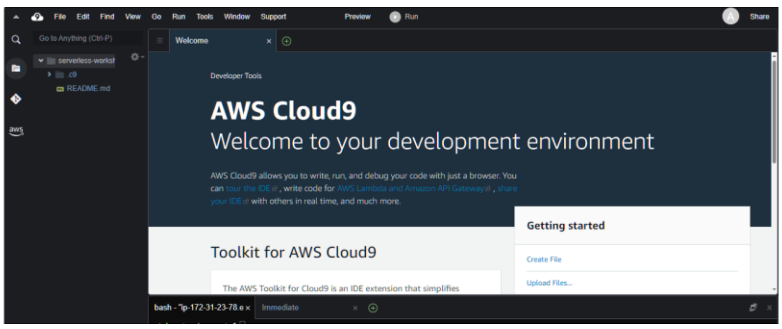
Environments (1)

	Name	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN
○	serverless-workshop	Open	EC2 instance	AWS Systems Manager (SSM)	Owner	arn:aws:iam::975049895916:user/admin

1. When it comes up, customize the environment by closing the **welcome tab** and **lower work area**, and opening a new terminal tab in the main work area:



Press Alt-T to open a new terminal at any time.

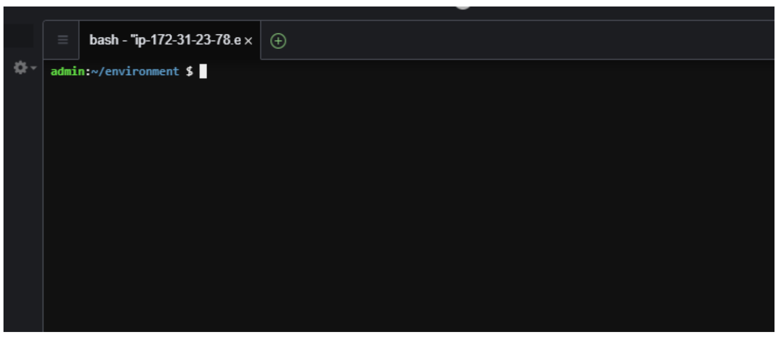


AWS Cloud9
Welcome to your development environment

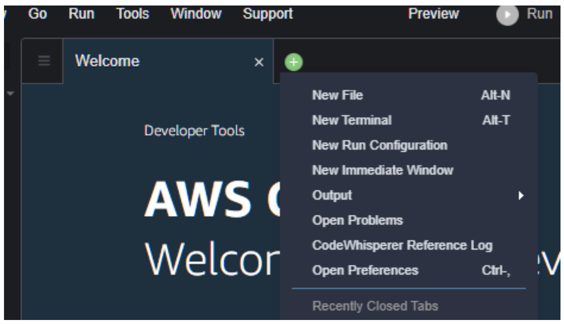
AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share your IDE with others in real time, and much more.

Toolkit for AWS Cloud9

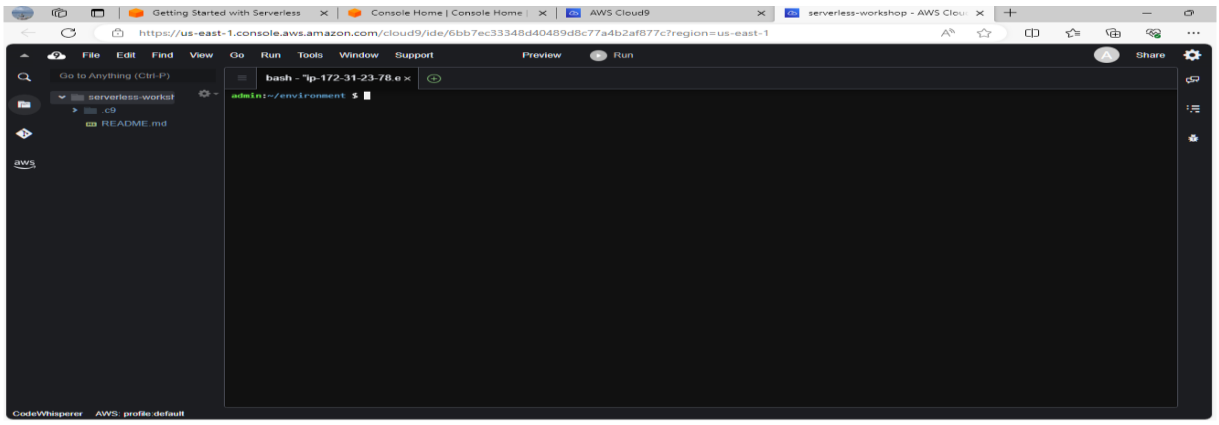
The AWS Toolkit for Cloud9 is an IDE extension that simplifies



```
admin:~/environment $
```



Your workspace should now look like this:



```
admin:~/environment $
```

TOTAL THERE ARE 6 STEPS INVOLVED TO COMPLETE THIS SERVERLESS PROJECT :

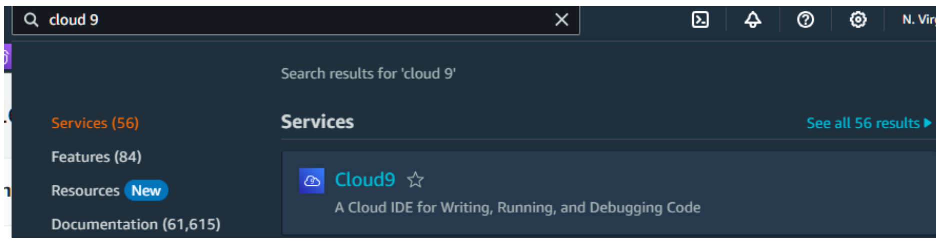
- Step 0: Prerequisites
- Step 1: Build a serverless backend: AWS Lambda, AWS SAM
- Step 2: Configure API authorization: API Gateway
- Step 3: Build and deploy a web application: AWS Amplify
- Step 4: Test the application
- Step 5: Configure image metadata extraction: Amazon Rekognition
- Step 6: Terminate Resources
- Conclusion:

Step 0: Prerequisites:

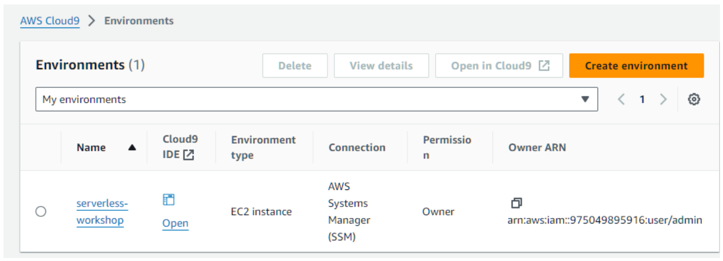
Open the Cloud 9 IDE

If you are part of an AWS-hosted event, the Cloud9 environment should already be set up in your AWS account.

1. Navigate to Cloud9 in the AWS Console (type Cloud9 in the services search bar)



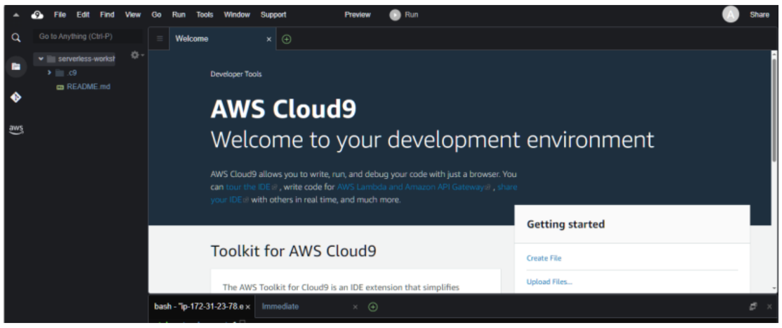
2. Choose the environment and click the Open button under Cloud 9 IDE.



Environments (1)

	Name	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN
○	serverless-workshop	Open	EC2 instance	AWS Systems Manager (SSM)	Owner	arn:aws:iam::975049895916:user/admin

3. Observe the Cloud 9 IDE has loaded



AWS Cloud9

Welcome to your development environment

AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share your IDE with others in real time, and much more.

Toolkit for AWS Cloud9

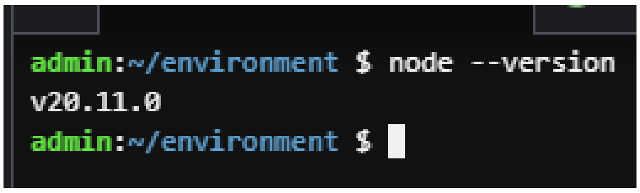
The AWS Toolkit for Cloud9 is an IDE extension that simplifies

Ensure Node.js Version

Cloud9 preinstalls Node.js, but it might be a version which is not compatible with this workshop. To ensure compatibility, we recommend you first check the current version, and install Node.js v20 (codename "Iron") only when required.
You can confirm the version you have installed by running the following command:

node –version

You should see output like this:
v20.10.0



```
admin:~/environment $ node --version
v20.11.0
admin:~/environment $ 
```

Clone the Git repository

In a Cloud9 terminal, run the following command:
cd ~/environment

git clone https://github.com/aws-samples/serverless-tasks-webapp

Avoiding free space issues with Cloud9

First, we need to ensure we're using the latest version of the AWS CLI:
sudo yum update -y
wget https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-linux-x86_64.zip
unzip aws-sam-cli-linux-x86_64.zip -d sam-installation

```
sudo ./sam-installation/install --update
sam –version
```

You will see output like this:

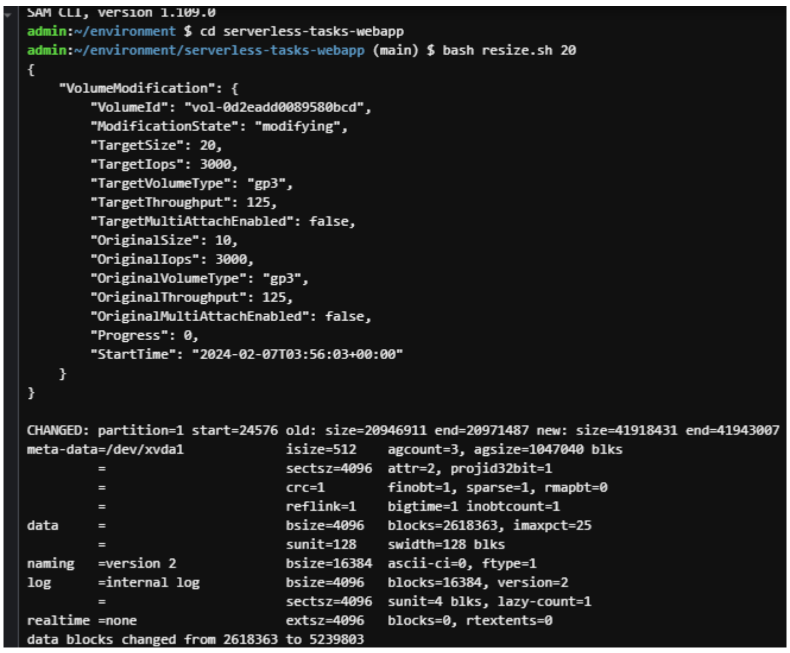
```
SAM CLI, version 1.109.0
```

Change to the repository directory:

```
cd serverless-tasks-webapp
```

Then run the following command, replacing 20 with the size in GiB you want to resize the Amazon EBS volume to:

```
bash resize.sh 20
```

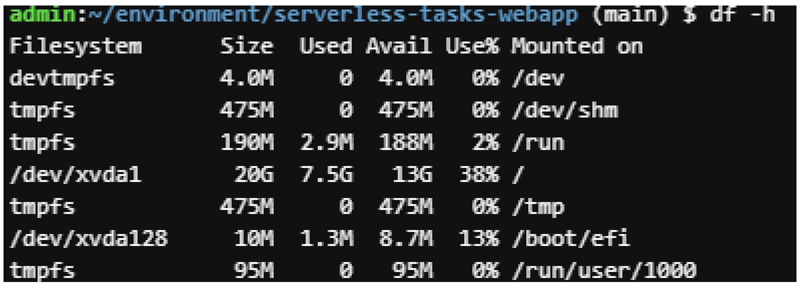


```
SAM CLI, version 1.109.0
admin:~/environment $ cd serverless-tasks-webapp
admin:~/environment/serverless-tasks-webapp (main) $ bash resize.sh 20
{
    "VolumeModification": {
        "VolumeId": "vol-0d2eadd0089580bcd",
        "ModificationState": "modifying",
        "TargetSize": 20,
        "TargetIops": 3000,
        "TargetVolumeType": "gp3",
        "TargetThroughput": 125,
        "TargetMultiAttachEnabled": false,
        "OriginalSize": 10,
        "OriginalIops": 3000,
        "OriginalVolumeType": "gp3",
        "OriginalThroughput": 125,
        "OriginalMultiAttachEnabled": false,
        "Progress": 0,
        "StartTime": "2024-02-07T03:56:03+00:00"
    }
}

CHANGED: partition=1 start=24576 old: size=20946911 end=20971487 new: size=41918431 end=41943007
meta-data=/dev/xvda1             isize=512    agcount=3, agsize=1047040 blks
         =                       sectsz=4096  attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1    bigtime=1 inobtcount=1
data     =                       bsize=4096   blocks=2618363, imaxpct=25
         =                       sunit=128    swidth=128 blks
naming   =version 2              bsize=16384  ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=16384, version=2
         =                       sectsz=4096  sunit=4 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
data blocks changed from 2618363 to 5239803
```

Now you can validate the new volume size by running the following command:

```
df -h
```

You should see something like this:



```
admin:~/environment/serverless-tasks-webapp (main) $ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           475M     0  475M   0% /dev/shm
tmpfs           190M  2.9M  188M   2% /run
/dev/xvda1       20G  7.5G   13G  38% /
tmpfs           475M     0  475M   0% /tmp
/dev/xvda128     10M  1.3M  8.7M  13% /boot/efi
tmpfs            95M     0   95M   0% /run/user/1000
```

The filesystem at `/dev/nvme0n1p1` now shows 13GB available, which is sufficient to complete this workshop.

Install Amplify CLI

The Amplify Command Line Interface (CLI) is a unified toolchain to create, integrate, and manage the AWS cloud services for your app.
The CLI uses AWS CloudFormation and nested stacks to allow you to add or modify configurations locally before you push them for execution in your account.
The CLI can configure a number of AWS services, including:

- auth (Amazon Cognito)
- storage (Amazon S3 & Amazon DynamoDB)
- function (AWS Lambda)
- api (AWS AppSync & Amazon API Gateway)
- analytics (Amazon Pinpoint)
- hosting (Amazon S3 and Amazon CloudFront distribution)
- notifications (Amazon Pinpoint)
- interactions (Amazon Lex)
- predictions (Amazon Rekognition, Amazon Textract, Amazon Translate, Amazon Polly, Amazon Transcribe, Amazon Comprehend, and Amazon SageMaker)

In this workshop, we will be configuring only hosting . AWS Amplify offers a fully managed service for deploying and hosting static web applications globally. This uses Amazon S3 and Amazon CloudFront to serve your static content.
Let's go ahead and install the Amplify CLI.

Follow the instructions

Install the latest version of the Amplify CLI, and a required dependency:

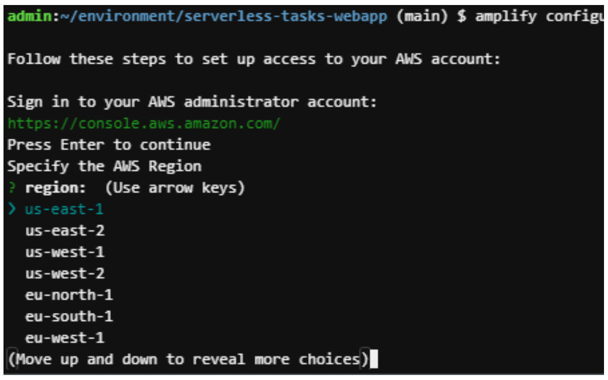
```
npm i -g @aws-amplify/cli
sudo yum -y install xdg-utils
```

Now it's time to setup the Amplify CLI. Configure Amplify by running the following command:

```
amplify configure
```

amplify configure will ask you to sign into the AWS Console. vIt will provide you a URL like to the console. Follow that link to create a user with **AdministratorAccess-Amplify** to your account to provision AWS resources for you.

Once you're signed in, Amplify CLI will ask you to create an IAM user and will prepoluate the name and permissions. Confirm that the permission policy applied is the **AdministratorAccess-Amplify** policy. Finish the user creation then copy the Access Key ID and Secret Access Keys.



```
admin:~/environment/serverless-tasks-webapp (main) $ amplify configu

Follow these steps to set up access to your AWS account:

Sign in to your AWS administrator account:
https://console.aws.amazon.com/
Press Enter to continue
Specify the AWS Region
? region:  (Use arrow keys)
❯ us-east-1
  us-east-2
  us-west-1
  us-west-2
  eu-north-1
  eu-south-1
  eu-west-1
(Move up and down to reveal more choices)
```

Once the user is created, Amplify CLI will ask you to provide the **accessKeyId** and the **secretAccessKey** to connect Amplify CLI with your newly created IAM user.

Enter the access key of the newly created user:

? accessKeyId: AKIA6GBMAV7WF5E6PZGN

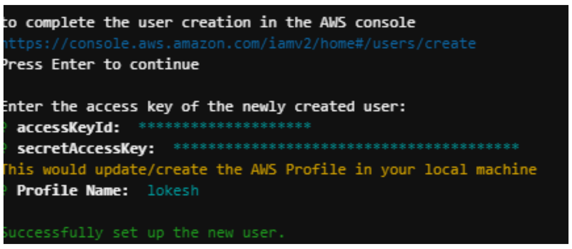
? secretAccessKey: peK7LyvJDcVCcxqepqI8KiX0B2OUlr7DSmAEEyU+

This would update/create the AWS Profile in your local machine

? Profile Name: lokesh

Successfully set up the new user.

Now you're ready to use Amplify!

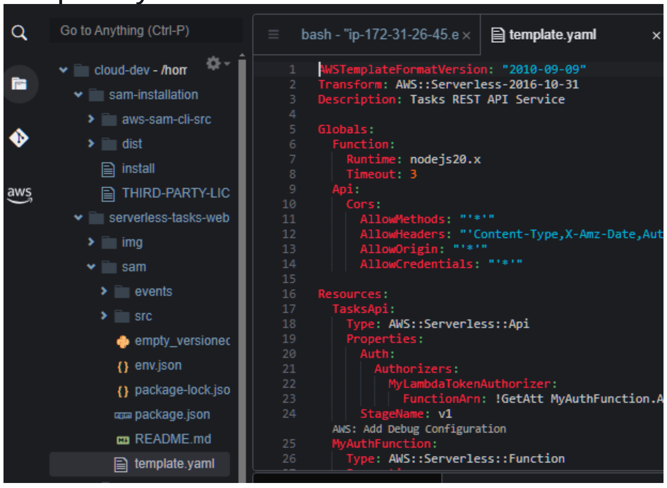


```
to complete the user creation in the AWS console
https://console.aws.amazon.com/iamv2/home#/users/create
Press Enter to continue

Enter the access key of the newly created user:
? accessKeyId:  ********************
? secretAccessKey:  ****************************************
This would update/create the AWS Profile in your local machine
? Profile Name:  lokesh

Successfully set up the new user.
```

Step 1: Build a serverless backend: AWS Lambda, AWS SAM

Backend starts with DynamoDb:

1. Go to Cloud 9 Terminal use the following commands :
   Ls
   Cd sam
2. You can click on sam installation on the left side of terminal , then go to template.yaml file.



```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31
Description: Tasks REST API Service

Globals:
  Function:
    Runtime: nodejs20.x
    Timeout: 3
  Api:
    Cors:
      AllowMethods: "'*'"
      AllowHeaders: "'Content-Type,X-Amz-Date,Aut
      AllowOrigin: "'*'"
      AllowCredentials: "'*'"

Resources:
  TasksApi:
    Type: AWS::Serverless::Api
    Properties:
      Auth:
        Authorizers:
          MyLambdaTokenAuthorizer:
            FunctionArn: !GetAtt MyAuthFunction.A
      StageName: v1
  AWS: Add Debug Configuration
  MyAuthFunction:
    Type: AWS::Serverless::Function
```

Define a DynamoDB table

What is Amazon DynamoDB?

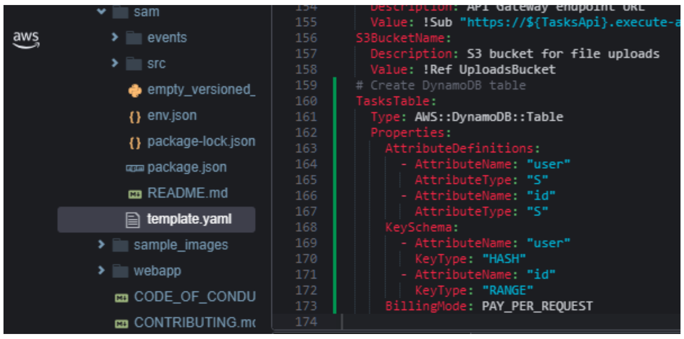
Amazon DynamoDB is a serverless key-value and document database that delivers single-digit millisecond performance at any scale.

Similar to other database systems, Amazon DynamoDB stores data in tables. In our sample application, we will store information about our tasks in a DynamoDB table. This table will be accessed by a Lambda function, in response to API calls from the web application.

We can define a DynamoDB table using SAM.

Add the DynamoDB table to the SAM template

The **AWS::DynamoDB::Table** resource is used to define the DynamoDB table.
Edit the file named **sam/template.yaml** and add the following code to the `Resources` section.



```yaml
    Value: !Sub "https://${TasksApi}.execute-a
  S3BucketName:
    Description: S3 bucket for file uploads
    Value: !Ref UploadsBucket
# Create DynamoDB table
TasksTable:
  Type: AWS::DynamoDB::Table
  Properties:
    AttributeDefinitions:
      - AttributeName: "user"
        AttributeType: "S"
      - AttributeName: "id"
        AttributeType: "S"
    KeySchema:
      - AttributeName: "user"
        KeyType: "HASH"
      - AttributeName: "id"
        KeyType: "RANGE"
    BillingMode: PAY_PER_REQUEST
```

Create a Lambda function

What is AWS Lambda?

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. Lambda automatically allocates compute power and runs your code based on the incoming request or event, for any scale of traffic.

How it works:

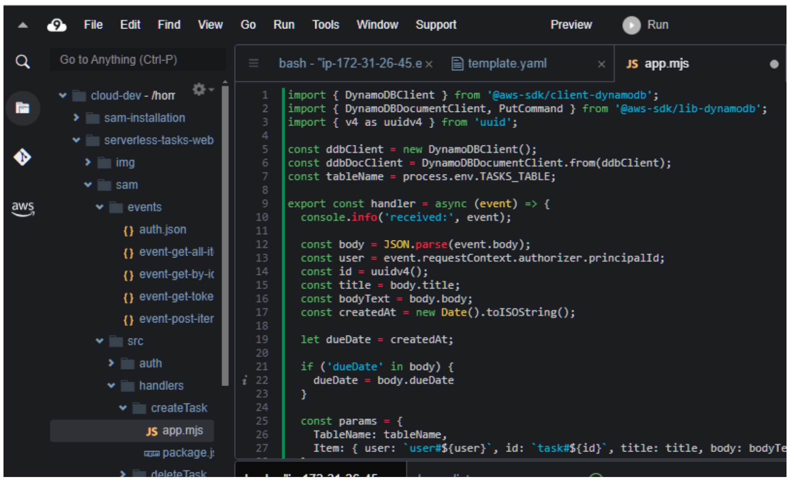
1. Upload your code to AWS Lambda or write code in Lambda's code editor. (In this workshop, we'll be writing code and uploading it using SAM.)
2. Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity.
3. Lambda runs your code only when triggered, using only the compute resources needed.
4. Just pay for the compute time you use.

Anatomy of a Lambda function

The Lambda function **handler** is the method in your function code that processes events. When your function is invoked, Lambda runs the handler method. When the handler exits or returns a response, it becomes available to handle another event.

Create a Lambda function

Go to terminal in the left side you'll find the file named **sam/src/handlers/createTask/app.mjs** and paste the following code into it:



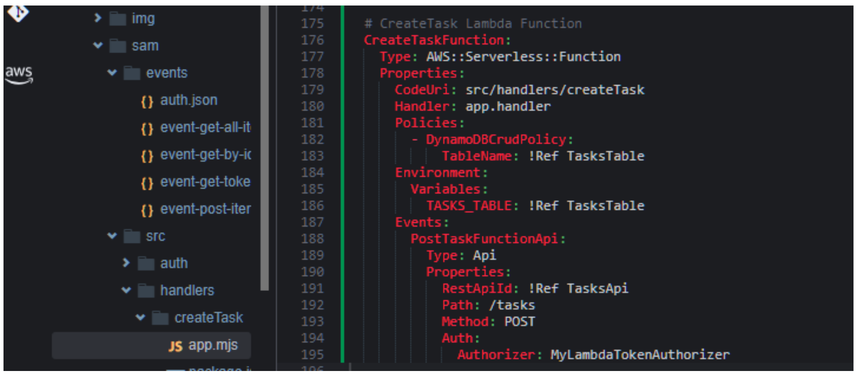
```javascript
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocumentClient, PutCommand } from '@aws-sdk/lib-dynamodb';
import { v4 as uuidv4 } from 'uuid';

const ddbClient = new DynamoDBClient();
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);
const tableName = process.env.TASKS_TABLE;

export const handler = async (event) => {
  console.info('received:', event);

  const body = JSON.parse(event.body);
  const user = event.requestContext.authorizer.principalId;
  const id = uuidv4();
  const title = body.title;
  const bodyText = body.body;
  const createdAt = new Date().toISOString();

  let dueDate = createdAt;

  if ('dueDate' in body) {
    dueDate = body.dueDate
  }

  const params = {
    TableName: tableName,
    Item: { user: `user#${user}`, id: `task#${id}`, title: title, body: bodyTe
```

Once you install **SAM PACKAGE** you'll get all these files.

Add the Lambda function to the SAM template

Add the following code to the **Resources** section of the **template.yml** file.

The **AWS::Serverless::Function** resource is used to define the Lambda function. The **CodeUri** property is used to specify the location of the **src/handlers/createTask/app.mjs** file.



```yaml
# CreateTask Lambda Function
CreateTaskFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: src/handlers/createTask
    Handler: app.handler
    Policies:
      - DynamoDBCrudPolicy:
          TableName: !Ref TasksTable
    Environment:
      Variables:
        TASKS_TABLE: !Ref TasksTable
    Events:
      PostTaskFunctionApi:
        Type: Api
        Properties:
          RestApiId: !Ref TasksApi
          Path: /tasks
          Method: POST
          Auth:
            Authorizer: MyLambdaTokenAuthorizer
```

Here we created & added one **DynamoDB** table & **Lambda Function** as shown in above snap.

Step 2: Configure API authorization: API Gateway

What is API GATEWAY ?

In AWS (Amazon Web Services), the API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

Here's what it does in simple terms:

1. **API Creation**: It allows you to create APIs quickly without having to worry about the underlying infrastructure.
2. **API Management**: It provides tools to manage APIs, such as versioning, documentation generation, and monitoring.
3. **Authorization and Authentication**: It enables you to control access to your APIs by implementing various authentication and authorization mechanisms.
4. **Traffic Control**: It helps in controlling the traffic to your APIs by providing features like throttling, caching, and request/response transformations.
5. **Integration**: It allows you to integrate your APIs with backend services hosted on AWS or elsewhere.
6. **Monitoring and Logging**: It provides monitoring and logging capabilities, allowing you to track the usage and performance of your APIs.

What is a Lambda Authorizer?

A Lambda authorizer is an API Gateway feature that uses a Lambda function to control access to your API. It is a way to add additional security to your API.

How does a Lambda Authorizer work?

When a client makes a request to one of your API's methods, API Gateway calls your Lambda authorizer, which takes the caller's identity as input and returns an IAM policy as output.

API Gateway supports open standards for authentication strategies such as OAuth and SAML .

Typically, an identity provider is used to authenticate the caller. The identity provider is responsible for verifying the caller's identity and returning the caller's identity. The caller's identity is then passed to the Lambda authorizer, which is responsible for controlling access to your API.

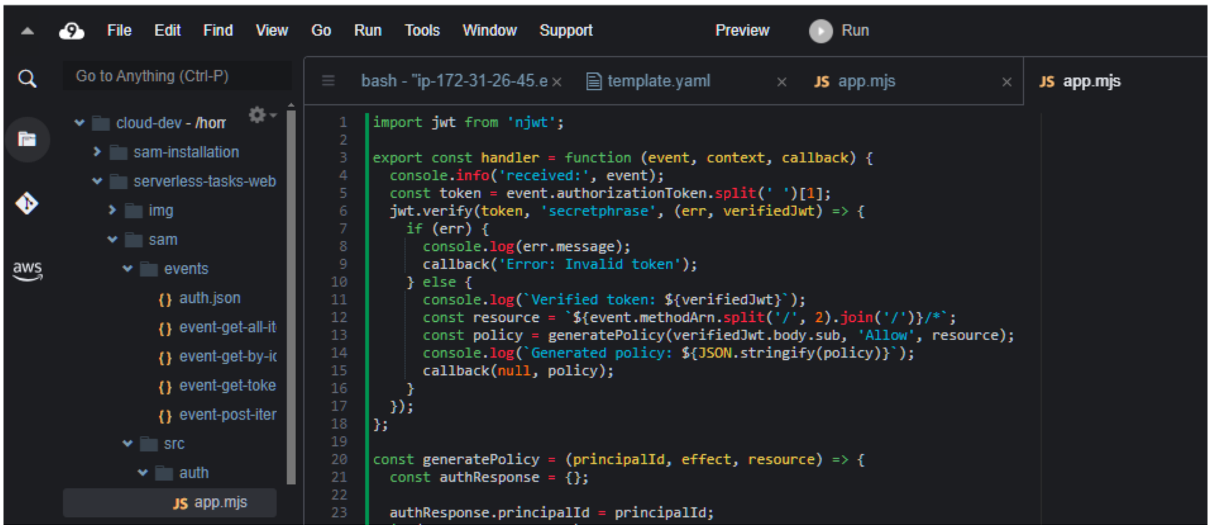
In this workshop, we won't be implementing the the identity provider needed to vend security tokens, so we will instead be mocking the identity provider. More on this later.

JSON Web Tokens (JWT)

The caller's identity is represented by a JSON Web Token (JWT) . A JWT is a compact, self-contained, signed JSON object. The JWT is typically used to represent an authenticated user's identity.

Configure a Lambda Authorizer

Go to the file named **sam/src/auth/app.mjs** and paste the following code into it:



```
import jwt from 'njwt';

export const handler = function (event, context, callback) {
  console.info('received:', event);
  const token = event.authorizationToken.split(' ')[1];
  jwt.verify(token, 'secretphrase', (err, verifiedJwt) => {
    if (err) {
      console.log(err.message);
      callback('Error: Invalid token');
    } else {
      console.log(`Verified token: ${verifiedJwt}`);
      const resource = `${event.methodArn.split('/', 2).join('/')}/*`;
      const policy = generatePolicy(verifiedJwt.body.sub, 'Allow', resource);
      console.log(`Generated policy: ${JSON.stringify(policy)}`);
      callback(null, policy);
    }
  });
};

const generatePolicy = (principalId, effect, resource) => {
  const authResponse = {};

  authResponse.principalId = principalId;
```

This is a simple token-based authorizer that demonstrates how to use an authorization token to allow a request. The token is passed in the **Authorization** header.

We then verify the token using the passphrase **secretphrase**. This is a well-known passphrase, shared with the web application. This is simply for demonstration purposes. We are simply mocking the identity provider here. In a real-world application, you would use a robust identity provider, such as Amazon Cognito, to verify the caller's identity.

We then generate an IAM policy that allows the caller to access the API. Note that this example generates an **Allow** policy for **any valid JWT token**.

Step 3: Build and deploy a web application: AWS Amplify

Overview

In this step, you'll build and the deploy the serverless application stack using SAM. You'll then configure the web application to communicate with an API Gateway endpoint. Then, you'll build the web application, and configure AWS Amplify to host the static resources for your web application.

Deploy the Serverless Application

In this step, we will use the SAM CLI to build and deploy the serverless application stack.

First, we need to ensure we're using the latest version of the SAM CLI:

```
sudo yum update -y
wget https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-linux-x86_64.zip
unzip aws-sam-cli-linux-x86_64.zip -d sam-installation
sudo ./sam-installation/install –update
```

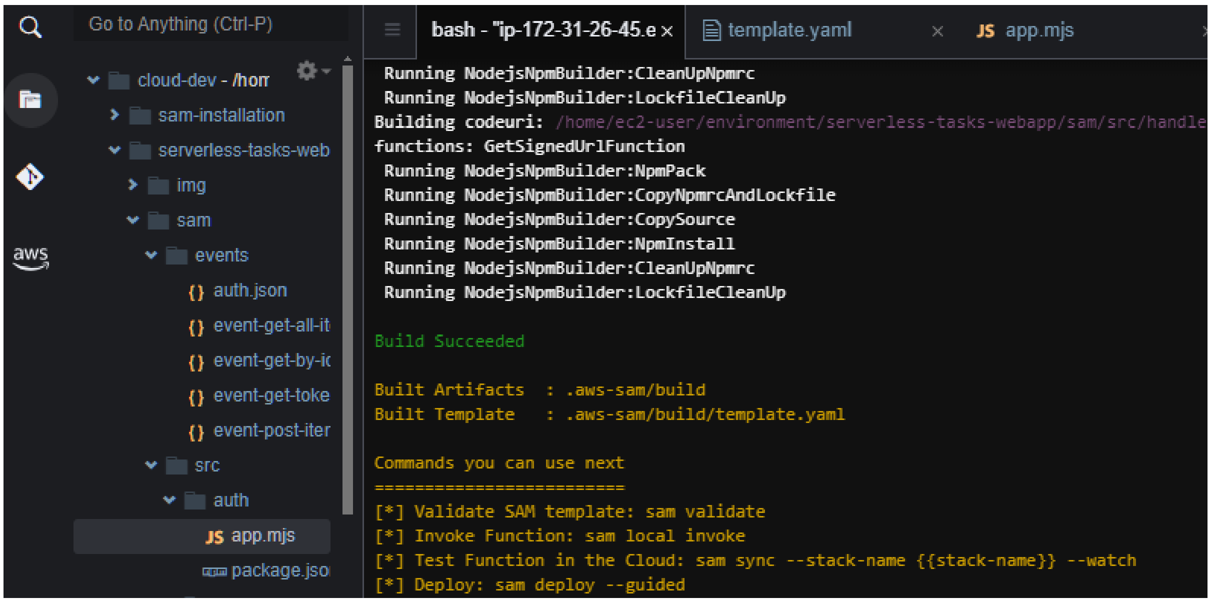
Make sure you're in the `sam` directory:

```
cd ~/environment/serverless-tasks-webapp/sam
```

Run the following command to build the application:

```
sam build
```

The **sam build** command processes your AWS SAM template file, application code, and any applicable language-specific files and dependencies. The command also copies build artifacts in the format and location expected for subsequent steps in your workflow.



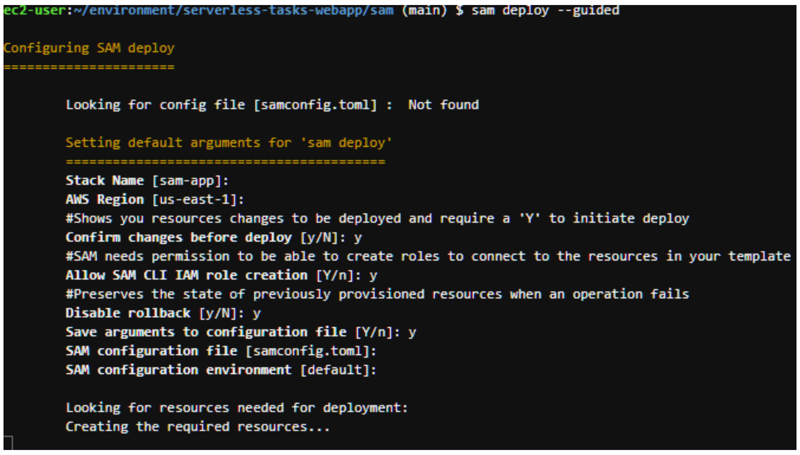
```
 Running NodejsNpmBuilder:CleanUpNpmrc
 Running NodejsNpmBuilder:LockfileCleanUp
Building codeuri: /home/ec2-user/environment/serverless-tasks-webapp/sam/src/handle
functions: GetSignedUrlFunction
 Running NodejsNpmBuilder:NpmPack
 Running NodejsNpmBuilder:CopyNpmrcAndLockfile
 Running NodejsNpmBuilder:CopySource
 Running NodejsNpmBuilder:NpmInstall
 Running NodejsNpmBuilder:CleanUpNpmrc
 Running NodejsNpmBuilder:LockfileCleanUp

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
```

Now we can deploy the application:

sam deploy –guided

This command packages and deploys the application to your AWS account. It provides a series of prompts:

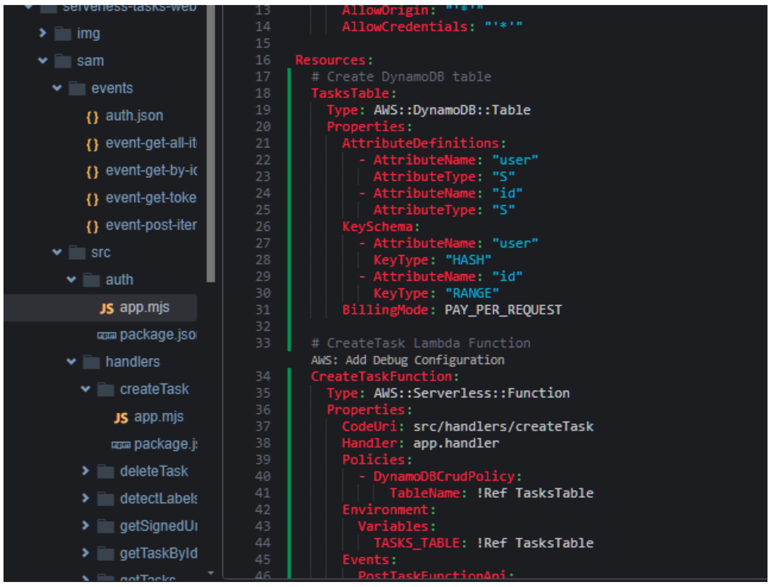
- **Stack Name:** The name of the stack to deploy to AWS CloudFormation. This should be unique to your account and Region. This workshop uses **tasks-app** throughout this project.
- **AWS Region:** The AWS Region you want to deploy your app to.
- **Confirm changes before deploy:** If set to yes, any change sets will be shown to you for manual review. If set to no, the AWS SAM CLI automatically deploys application changes.



```
ec2-user:~/environment/serverless-tasks-webapp/sam (main) $ sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Not found

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app]:
        AWS Region [us-east-1]:
        #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
        Confirm changes before deploy [y/N]: y
        #SAM needs permission to be able to create roles to connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]: y
        #Preserves the state of previously provisioned resources when an operation fails
        Disable rollback [y/N]: y
        Save arguments to configuration file [Y/n]: y
        SAM configuration file [samconfig.toml]:
        SAM configuration environment [default]:

        Looking for resources needed for deployment:
        Creating the required resources...
```

You may respond to the remaining prompts with the default options.

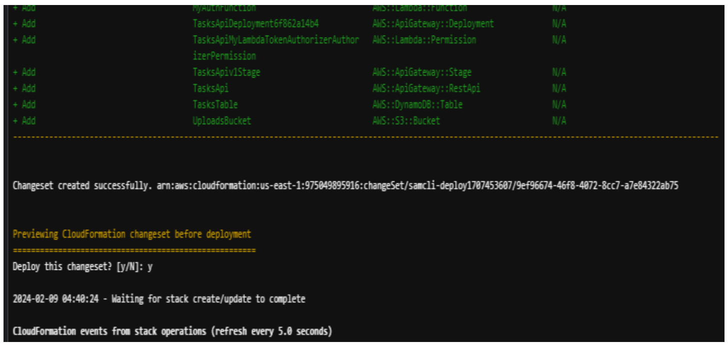
Here you'll detect an error displaying that status as failed . to rectify this error , then you have to modify some changes in template.yaml in main src file as shown in below snap.



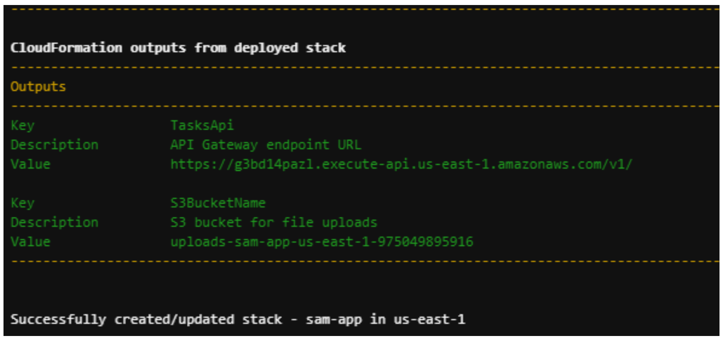
```
          AllowOrigin: "'*'"
          AllowCredentials: "'*'"

  Resources:
    # Create DynamoDB table
    TasksTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:
          - AttributeName: "user"
            AttributeType: "S"
          - AttributeName: "id"
            AttributeType: "S"
        KeySchema:
          - AttributeName: "user"
            KeyType: "HASH"
          - AttributeName: "id"
            KeyType: "RANGE"
        BillingMode: PAY_PER_REQUEST

    # CreateTask Lambda Function
    AWS: Add Debug Configuration
    CreateTaskFunction:
      Type: AWS::Serverless::Function
      Properties:
        CodeUri: src/handlers/createTask
        Handler: app.handler
        Policies:
          - DynamoDBCrudPolicy:
              TableName: !Ref TasksTable
        Environment:
          Variables:
            TASKS_TABLE: !Ref TasksTable
        Events:
          PostTaskFunctionApi:
```

Before we added dynamodb & lambda function at the end of template.yaml file , now we have to replace it in resources place as shown in above snap.

After replacing run the same commands & you'll see the output like this :

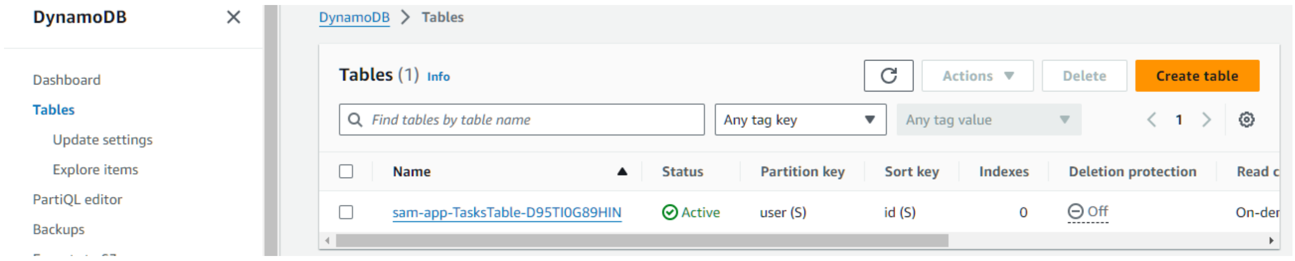


```
+ Add                   MyAuthFunction                          AWS::Lambda::Function           N/A
+ Add                   TasksApiDeployment6f862a14b4            AWS::ApiGateway::Deployment     N/A
+ Add                   TasksApiMyLambdaTokenAuthorizerAuthor   AWS::Lambda::Permission         N/A
                        izerPermission
+ Add                   TasksApiv1Stage                         AWS::ApiGateway::Stage          N/A
+ Add                   TasksApi                                AWS::ApiGateway::RestApi        N/A
+ Add                   TasksTable                              AWS::DynamoDB::Table            N/A
+ Add                   UploadsBucket                           AWS::S3::Bucket                 N/A
-----------------------------------------------------------------------------------------------------


Changeset created successfully. arn:aws:cloudformation:us-east-1:975049895916:changeSet/samcli-deploy1707453607/9ef96674-46f8-4072-8cc7-a7e84322ab75


Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: y

2024-02-09 04:40:24 - Waiting for stack create/update to complete

CloudFormation events from stack operations (refresh every 5.0 seconds)
```



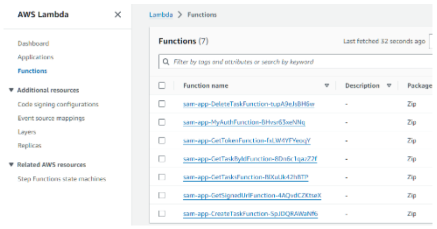
```
-----------------------------------------------------------------------------------------------
CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------------------------
Outputs
-----------------------------------------------------------------------------------------------
Key                 TasksApi
Description         API Gateway endpoint URL
Value               https://g3bd14pazl.execute-api.us-east-1.amazonaws.com/v1/

Key                 S3BucketName
Description         S3 bucket for file uploads
Value               uploads-sam-app-us-east-1-975049895916
-----------------------------------------------------------------------------------------------


Successfully created/updated stack - sam-app in us-east-1
```

For checking of database which you have created by using this serverless , go to AWS console , navigate **DYNAMOBD Dashboard** , in the section of **TABLES** you can see the **DYNAMODB TABLE** which you've created as shown in below snap.

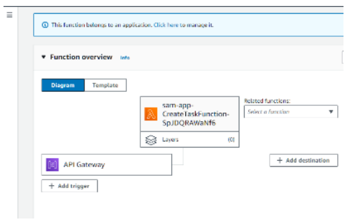


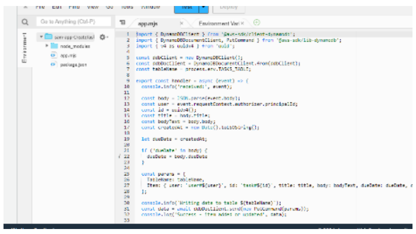
	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read c
☐	sam-app-TasksTable-D95TI0G89HIN	✓ Active	user (S)	id (S)	0	Off	On-der

For checking of **LAMBDA** which you have created by using this serverless , go to AWS console , navigate **LAMBDA Dashboard** , in the section of **FUNCTIONS** you can see the **LAMBDA FUNCTIONS** which you've created as shown in below snap.



	Function name	Description	Package
☐	sam-app-DeleteTaskFunction-tupA9eJsBH6w	-	Zip
☐	sam-app-MyAuthFunction-BHvsr63xeNNq	-	Zip
☐	sam-app-GetTokenFunction-fxLW4YFYeoqY	-	Zip
☐	sam-app-GetTaskByIdFunction-8Dn6c1qazZ2f	-	Zip
☐	sam-app-GetTasksFunction-BIXuUk42hBTP	-	Zip
☐	sam-app-GetSignedUrlFunction-4AQvdCZKtseX	-	Zip
☐	sam-app-CreateTaskFunction-SpJDQRAWaNf6	-	Zip





```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocumentClient, PutCommand } from '@aws-sdk/lib-dynamodb';
import { v4 as uuidv4 } from 'uuid';

const ddbClient = new DynamoDBClient();
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);
const tableName = process.env.TASKS_TABLE;

export const handler = async (event) => {
  console.info('received:', event);

  const body = JSON.parse(event.body);
  const user = event.requestContext.authorizer.principalId;
  const id = uuidv4();
  const title = body.title;
  const bodyText = body.body;
  const createdAt = new Date().toISOString();

  let dueDate = createdAt;

  if ('dueDate' in body) {
    dueDate = body.dueDate
  }

  const params = {
    TableName: tableName,
    Item: { user: `user#${user}`, id: `task#${id}`, title: title, body: bodyText, dueDate: dueDate, 
  };

  console.info(`Writing data to table ${tableName}`);
  const data = await ddbDocClient.send(new PutCommand(params));
  console.log('Success - item added or updated', data);

```

Congratulations! You've successfully deployed your serverless application to AWS.

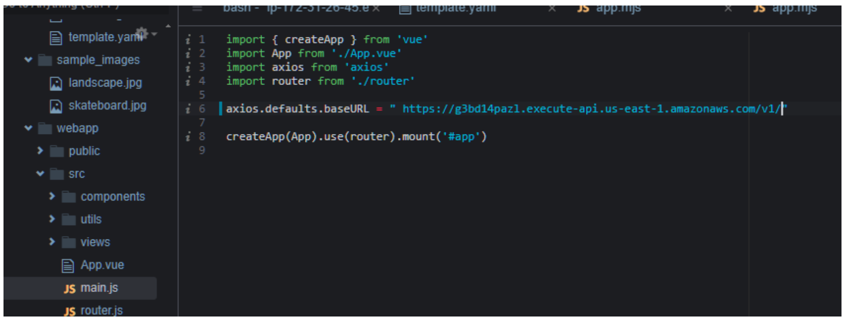
Configure the API Gateway endpoint

From the **sam deploy** output in a previous step, you can get the URL of the API Gateway endpoint. It should be of the form:

https://<api-gateway-name>.execute-api.<region>.amazonaws.com/<stage>

The web application requires a configuration for this endpoint so it knows where to send requests.

Open **/webapp/src/main.js** in an editor, and paste in the endpoint URL for the value of **axios.defaults.baseURL:**



```
import { createApp } from 'vue'
import App from './App.vue'
import axios from 'axios'
import router from './router'

axios.defaults.baseURL = " https://g3bd14pazl.execute-api.us-east-1.amazonaws.com/v1/"

createApp(App).use(router).mount('#app')

```

Build the web application

Ensure you are in the root of the web project (`/webapp`):

cd ~/environment/serverless-tasks-webapp/webapp

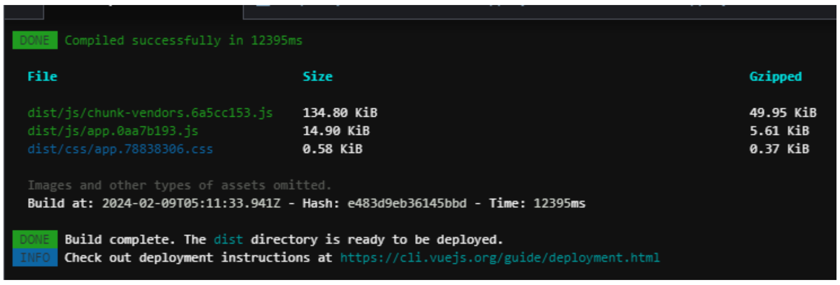
Install the dependencies:

npm install

Build the web application:

npm run build

You should see output like this:



```
 DONE  Compiled successfully in 12395ms

  File                                 Size                                                   Gzipped

  dist/js/chunk-vendors.6a5cc153.js    134.80 KiB                                             49.95 KiB
  dist/js/app.0aa7b193.js              14.90 KiB                                              5.61 KiB
  dist/css/app.78838306.css            0.58 KiB                                               0.37 KiB

  Images and other types of assets omitted.
  Build at: 2024-02-09T05:11:33.941Z - Hash: e483d9eb36145bbd - Time: 12395ms

 DONE  Build complete. The dist directory is ready to be deployed.
 INFO  Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html
```

Initialize Amplify

It's time to set up Amplify so that we can create the necessary backend services needed to support the app. In this step, you'll configure Amplify hosting.

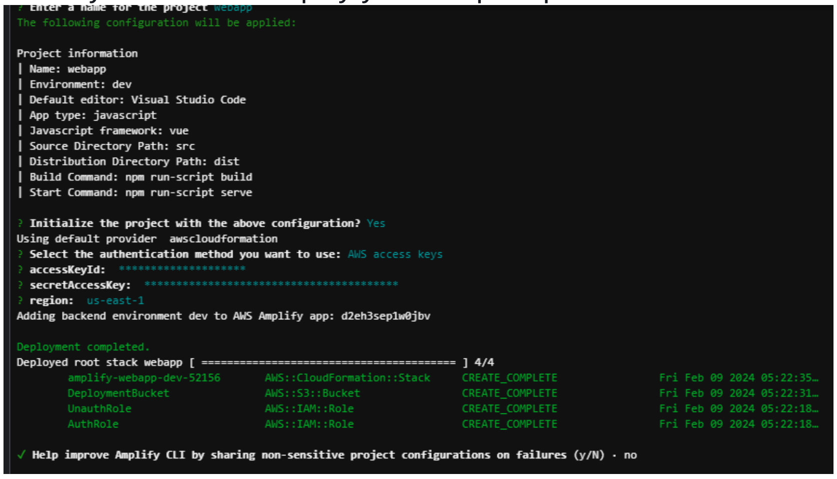
Ensure you are in the root of the web project (`/webapp`):

cd ~/environment/serverless-tasks-webapp/webapp

Initialize Amplify:

amplify init

When you initialize Amplify you'll be prompted for some information about the app:



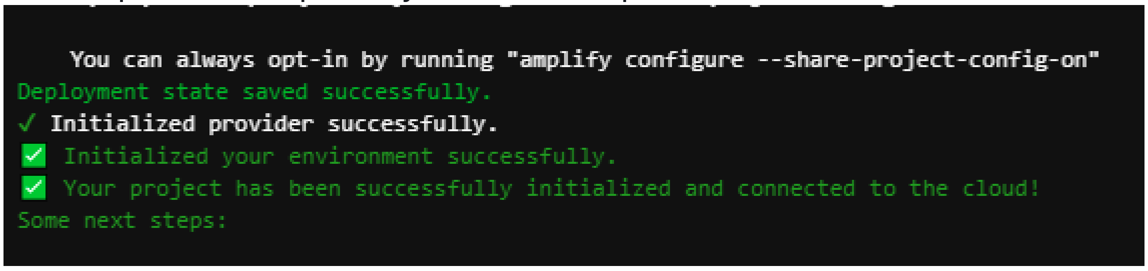
```
? Enter a name for the project webapp
The following configuration will be applied:

Project information
| Name: webapp
| Environment: dev
| Default editor: Visual Studio Code
| App type: javascript
| Javascript framework: vue
| Source Directory Path: src
| Distribution Directory Path: dist
| Build Command: npm run-script build
| Start Command: npm run-script serve

? Initialize the project with the above configuration? Yes
Using default provider  awscloudformation
? Select the authentication method you want to use: AWS access keys
? accessKeyId:  ********************
? secretAccessKey:  ****************************************
? region:  us-east-1
Adding backend environment dev to AWS Amplify app: d2eh3sep1w0jbv

Deployment completed.
Deployed root stack webapp [ ======================================== ] 4/4
        amplify-webapp-dev-52156       AWS::CloudFormation::Stack     CREATE_COMPLETE                Fri Feb 09 2024 05:22:35…
        DeploymentBucket               AWS::S3::Bucket                CREATE_COMPLETE                Fri Feb 09 2024 05:22:31…
        UnauthRole                     AWS::IAM::Role                 CREATE_COMPLETE                Fri Feb 09 2024 05:22:18…
        AuthRole                       AWS::IAM::Role                 CREATE_COMPLETE                Fri Feb 09 2024 05:22:18…

✓ Help improve Amplify CLI by sharing non-sensitive project configurations on failures (y/N) · no
```

When you initialize a new Amplify project, a few things happen:

- It creates a top level directory called **amplify** that stores your backend definition. During the workshop you'll add capabilities such as a Amplify hosting. As you add features, the **amplify** folder will grow with infrastructure-as-code templates that define your backend stack. Infrastructure-as-code is a best practice way to create a replicable backend stack.
- It creates a file called **aws-exports.js** in the **src** directory that holds all the configuration for the services you create with Amplify. This is how the Amplify client is able to get the necessary information about your backend services.
- It modifies the **.gitignore** file, adding some generated files to the ignore list
- A cloud project is created for you in the AWS Amplify Console that can be accessed by running **amplify console**. The Console provides a list of backend environments, deep links to provisioned resources per Amplify category,

status of recent deployments, and instructions on how to promote, clone, pull, and delete backend resources

When the process completes, you will see output like this:



```
    You can always opt-in by running "amplify configure --share-project-config-on"
Deployment state saved successfully.
✓ Initialized provider successfully.
✅ Initialized your environment successfully.
✅ Your project has been successfully initialized and connected to the cloud!
Some next steps:
```

Now that our Vue.js app is set up and Amplify is initialized, we're ready to add Amplify hosting in the next steps.

Add Amplify hosting

You've successfully built your first app with Amplify! Now that you've built something, it's time to deploy it to the web with Amplify Console!

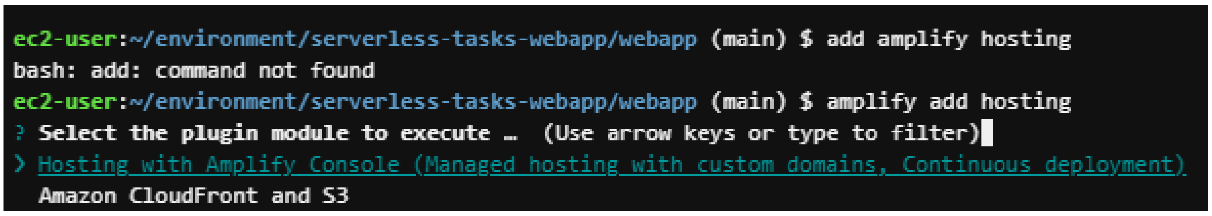
All of your static web content including HTML, CSS, JavaScript, images and other files will be managed by AWS Amplify Console. Your end users will then access your site using the public website URL exposed by AWS Amplify Console. You don't need to run any web servers or use other services in order to make your site available.

You can manually deploy your web app or setup automatic continuous deployment. In this workshop, we'll cover how to manually deploy and host your static web app to quickly share with others. If you want to learn about continuous deployment instead, please follow this guide .

Run the following command and select the **bolded options**.

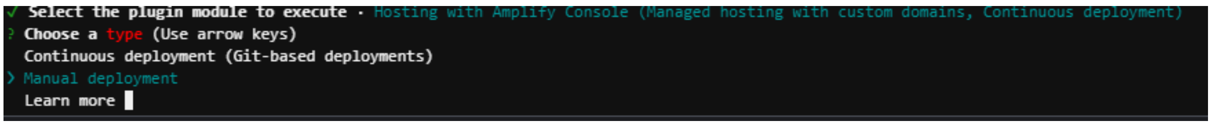
```
amplify add hosting
```

You will be prompted to select a plugin to use for hosting. Select **Hosting with Amplify Console**:



```
ec2-user:~/environment/serverless-tasks-webapp/webapp (main) $ add amplify hosting
bash: add: command not found
ec2-user:~/environment/serverless-tasks-webapp/webapp (main) $ amplify add hosting
? Select the plugin module to execute … (Use arrow keys or type to filter)
❯ Hosting with Amplify Console (Managed hosting with custom domains, Continuous deployment)
  Amazon CloudFront and S3
```

Next, choose deployment method. Select **Manual deployment**:



```
✓ Select the plugin module to execute · Hosting with Amplify Console (Managed hosting with custom domains, Continuous deployment)
? Choose a type (Use arrow keys)
  Continuous deployment (Git-based deployments)
❯ Manual deployment
  Learn more
```

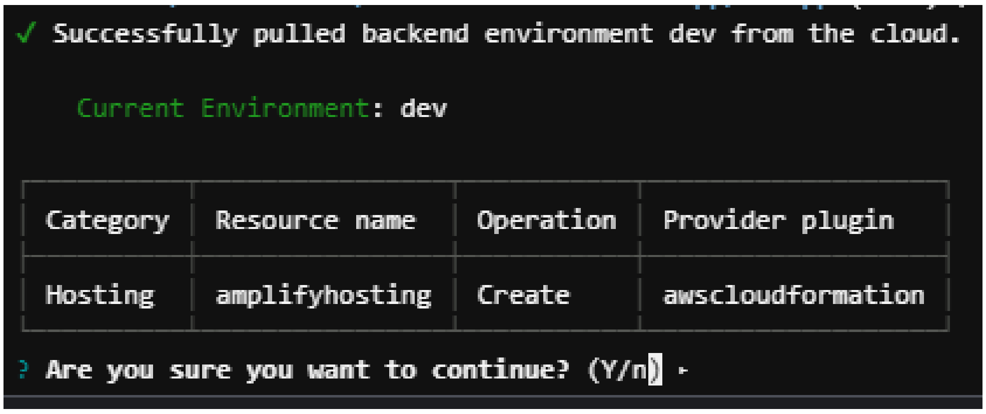
In the next step, we will publish your app to the web.

Deploy the application

Run the following command to publish your app.

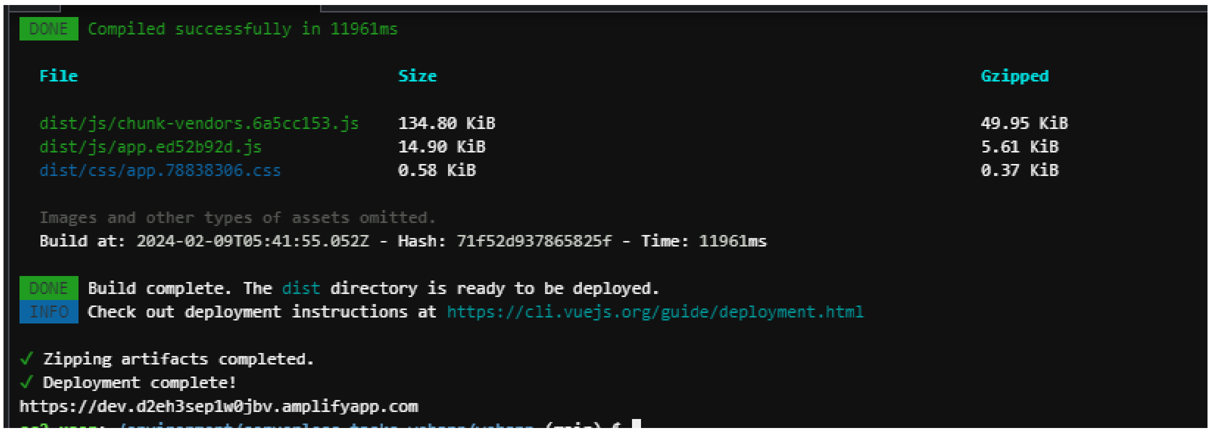
```
amplify publish
```

You may answer **Yes** to this prompt:



```
✓ Successfully pulled backend environment dev from the cloud.

    Current Environment: dev

┌──────────┬─────────────────┬───────────┬───────────────────┐
│ Category │ Resource name   │ Operation │ Provider plugin   │
├──────────┼─────────────────┼───────────┼───────────────────┤
│ Hosting  │ amplifyhosting  │ Create    │ awscloudformation │
└──────────┴─────────────────┴───────────┴───────────────────┘
? Are you sure you want to continue? (Y/n)
```

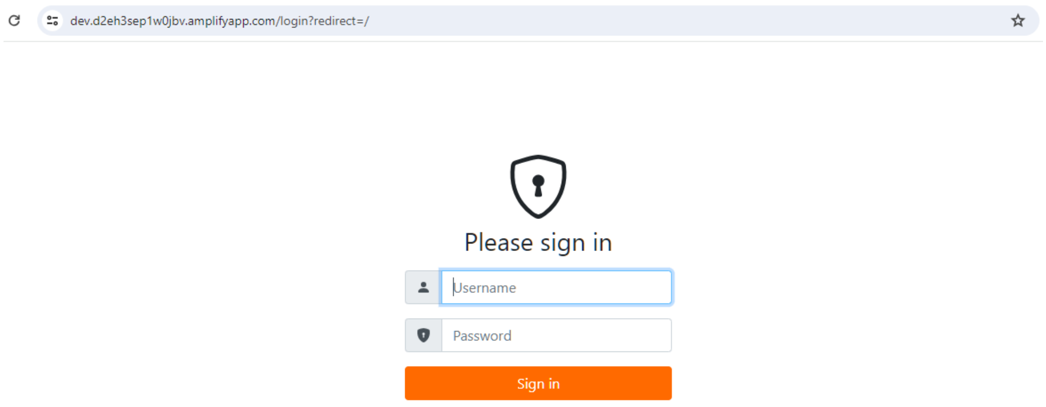
This will deploy a CloudFormation stack, and you will see output like the following:



```
 DONE  Compiled successfully in 11961ms

  File                                 Size                                    Gzipped

  dist/js/chunk-vendors.6a5cc153.js    134.80 KiB                              49.95 KiB
  dist/js/app.ed52b92d.js              14.90 KiB                               5.61 KiB
  dist/css/app.78838306.css            0.58 KiB                                0.37 KiB

  Images and other types of assets omitted.
  Build at: 2024-02-09T05:41:55.052Z - Hash: 71f52d937865825f - Time: 11961ms

 DONE  Build complete. The dist directory is ready to be deployed.
 INFO  Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html

✓ Zipping artifacts completed.
✓ Deployment complete!
https://dev.d2eh3sep1w0jbv.amplifyapp.com
```

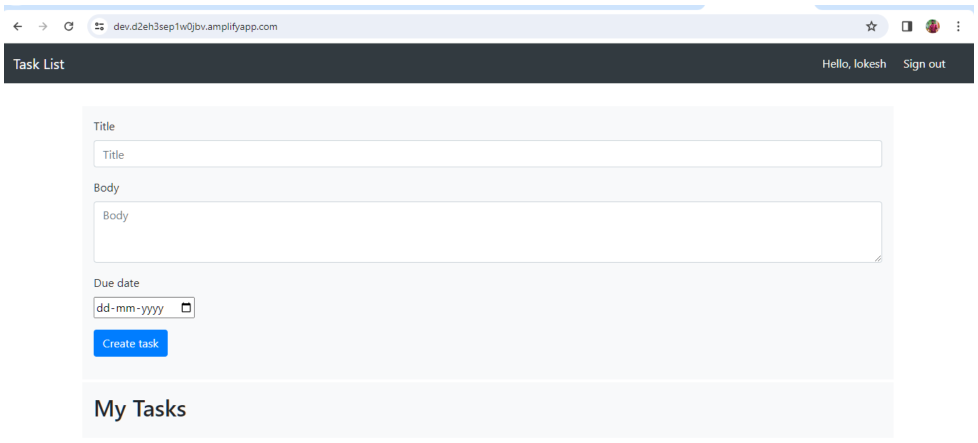
👏 Congratulations, your app is online!

After publishing, your terminal will display your app URL hosted on an **amplifyapp.com** domain.


dev.d2eh3sep1w0jbv.amplifyapp.com/login?redirect=/

Please sign in

Authentication
You may log in using **any username and password** you would like. There is **no authentication** backend configured.
The main application screen looks like this:

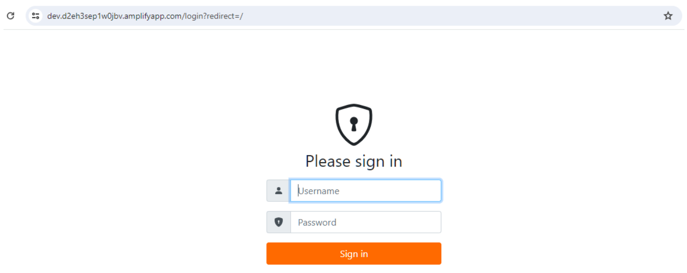


Whenever you have additional web application changes to publish, just re-run the **amplify publish** command. We will not be modifying the web application in this module, only the SAM template and Lambda functions.

To view your app and hosting configuration in the Amplify Console, run the **amplify console** command.

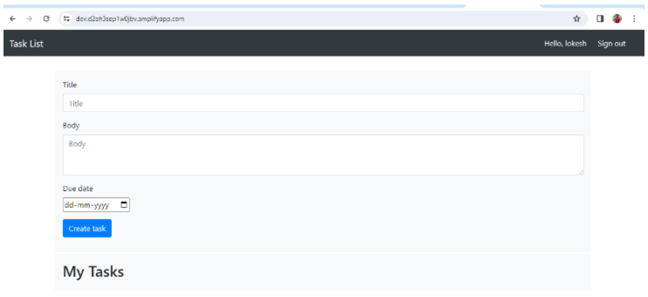
Step 4: Test the application

Visit your newly published application at the URL provided by the **deploy** step **(e.g. https://dev.xxxxxxxxxxxxxx.amplifyapp.com).**

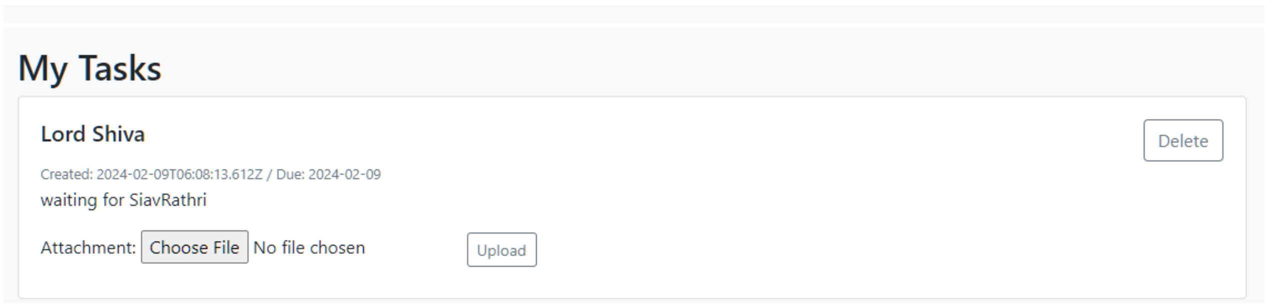


Since we are mocking authentication in this workshop, you may log in with any username and password you like.

The main application screen looks like this:



You can create a new task by providing a **title** and **body**. The **due date** is optional. Click **Create task** to create the new task. The new task will appear in the list of tasks.



My Tasks

Lord Shiva

Created: 2024-02-09T06:08:13.612Z / Due: 2024-02-09

waiting for SiavRathri

What just happened?

When you click **Create task**, the following happens:

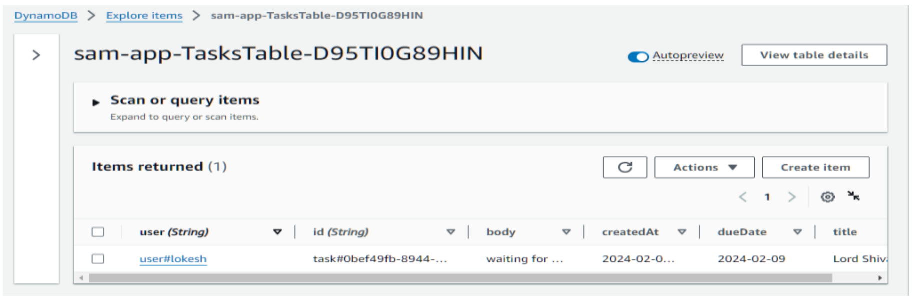
1. A **POST** request is sent to the /**tasks** API Gateway endpoint.
2. The request is authenticated by the Lambda authorizer, using the **Authentication: Bearer <token>** HTTP header.
3. The request body is forwarded to the **CreateTaskFunction** Lambda function.
4. The task data is stored in the DynamoDB table.
5. An HTTP 200 (success) response is returned to the web application.
6. The web application refreshes the list of tasks by performing a `GET` request to the **/tasks** endpoint.
7. Again, the request is authenticated by the Lambda authorizer, and the request is forwarded to the **GetTasksFunction** Lambda function.
8. The **GetTasksFunction** Lambda function retrieves the list of tasks for the authenticated user from the DynamoDB table.

   You may add as many tasks as you like. Try deleting a task and understand how the request flows through the application stack.

Viewing the data in DynamoDB

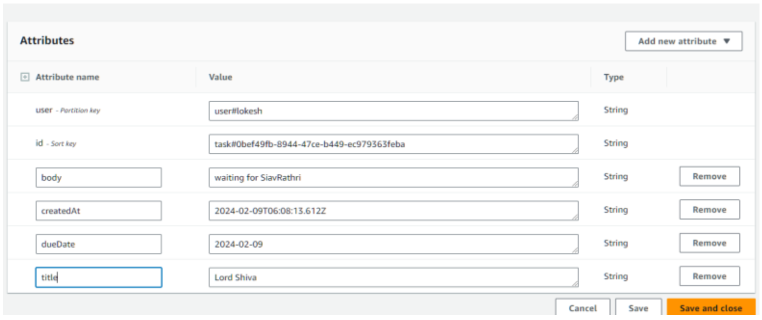
1. To view the data in DynamoDB, navigate to the DynamoDB console.
2. In the left navigation bar, select **Tables**.
3. Select the table whose name contains **tasks-app-TasksTable**. This is the table created by the SAM template.
4. Click the **Explore table items** button.

You will see the data in the table:


DynamoDB > Explore items > sam-app-TasksTable-D95TI0G89HIN
sam-app-TasksTable-D95TI0G89HIN
Autopreview
View table details
Scan or query items
Expand to query or scan items.
Items returned (1)
Actions
Create item

user (String)	id (String)	body	createdAt	dueDate	title
user#lokesh	task#0bef49fb-8944-...	waiting for ...	2024-02-0...	2024-02-09	Lord Shiv

Clicking the item's partition key (**user**) will take you to the item editor, where you can see the values in more detail, as well as edit them:



Attribute name	Value	Type	
user - Partition key	user#lokesh	String	
id - Sort key	task#0bef49fb-8944-47ce-b449-ec979363feba	String	
body	waiting for SiavRathri	String	Remove
createdAt	2024-02-09T06:08:13.612Z	String	Remove
dueDate	2024-02-09	String	Remove
title	Lord Shiva	String	Remove

Any changes you make to the item will be reflected in the web application upon refresh.

Logging and Monitoring

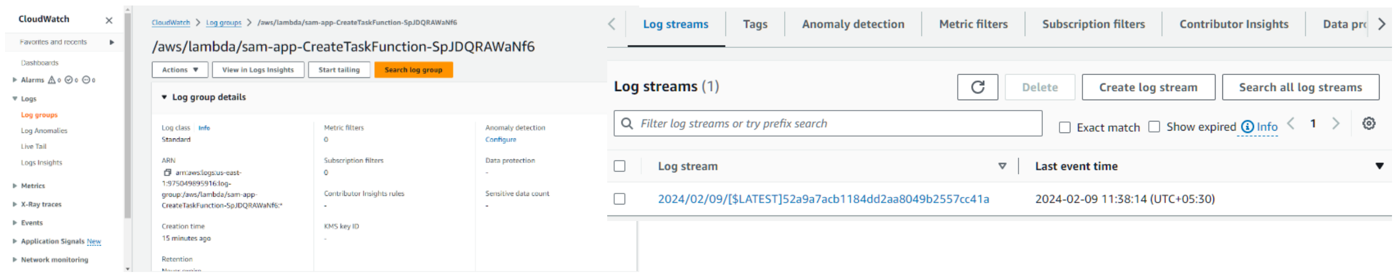
Logging

Centralized logging provides two important benefits. First, it places all of your log records in a single location and in a standardized format, greatly simplifying log analysis and correlation tasks. Second, it provides you with a secure storage area for your log data.

In AWS Lambda, the default logging service is Amazon CloudWatch.

Lambda automatically streams details about each function invocation, along with logs and other output from your function's code to CloudWatch Logs.

Log groups are a standard part of CloudWatch and used to organize all logging. Any log generated by a Lambda function uses the naming convention */aws/lambda/function-name*. A log group is a logical collection of log streams, which can you explore in the CloudWatch console:



/aws/lambda/sam-app-CreateTaskFunction-SpJDQRAWaNf6

Log group details

Log class: Standard
ARN: arn:aws:logs:us-east-1:975049895916:log-group:/aws/lambda/sam-app-CreateTaskFunction-SpJDQRAWaNf6:*
Creation time: 15 minutes ago
Retention
Metric filters: 0
Subscription filters: 0
Contributor Insights rules: -
KMS key ID: -
Anomaly detection: Configure
Data protection: -
Sensitive data count: -

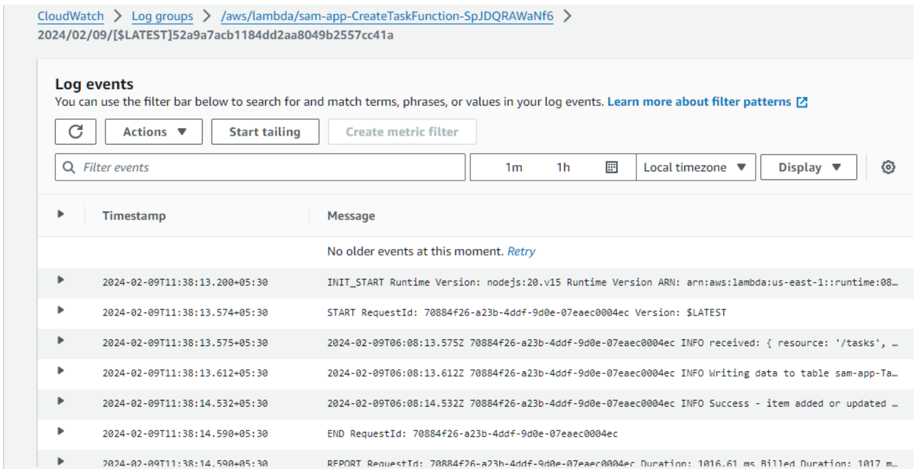
Log streams (1)

Log stream	Last event time
2024/02/09/[$LATEST]52a9a7acb1184dd2aa8049b2557cc41a	2024-02-09 11:38:14 (UTC+05:30)

Each instance of a Lambda function has a dedicated log stream. If a function scales up, each concurrent instance has its own log stream. Each time an execution environment is reaped and a new environment is created in response to an invocation, this generates a new log stream. The naming convention for log streams is:

YYYY/MM//DD [Function version] [Execution environment GUID]

A single execution environment writes to the same log stream during its lifetime.
For example, this is the log stream for the **GetTasksFunction** function:


CloudWatch > Log groups > /aws/lambda/sam-app-CreateTaskFunction-SpJDQRAWaNf6 >
Actions
Start tailing
Create metric filter
Filter events
1m
1h
Local timezone
Display

2024/02/09/[$LATEST]52a9a7acb1184dd2aa8049b2557cc41a

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                       Message
                                No older events at this moment. Retry
2024-02-09T11:38:13.200+05:30   INIT_START Runtime Version: nodejs:20.v15 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:08…
2024-02-09T11:38:13.574+05:30   START RequestId: 70884f26-a23b-4ddf-9d0e-07eaec0004ec Version: $LATEST
2024-02-09T11:38:13.575+05:30   2024-02-09T06:08:13.575Z 70884f26-a23b-4ddf-9d0e-07eaec0004ec INFO received: { resource: '/tasks', …
2024-02-09T11:38:13.612+05:30   2024-02-09T06:08:13.612Z 70884f26-a23b-4ddf-9d0e-07eaec0004ec INFO Writing data to table sam-app-Ta…
2024-02-09T11:38:14.532+05:30   2024-02-09T06:08:14.532Z 70884f26-a23b-4ddf-9d0e-07eaec0004ec INFO Success - item added or updated …
2024-02-09T11:38:14.590+05:30   END RequestId: 70884f26-a23b-4ddf-9d0e-07eaec0004ec
2024-02-09T11:38:14.590+05:30   REPORT RequestId: 70884f26-a23b-4ddf-9d0e-07eaec0004ec Duration: 1016.61 ms Billed Duration: 1017 m…
```

These logs show:

- **RequestId:** this is a unique ID generated per request. If the Lambda function retries a request, this ID does not change and appears in the logs for each subsequent retry.
- **Start/End:** these values bookmark a single invocation, so every log line between these belongs to the same invocation.
- **Duration:** the total invocation time for the handler function, excluding “INIT” code.
- **Billed Duration:** applies rounding logic for billing purposes.
- **Memory Size:** the amount of memory allocated to the function.
- **Max Memory Used:** the actual memory used during the invocation.
- **Init Duration:** the time taken to run the “INIT” section of code, outside of the main handler.

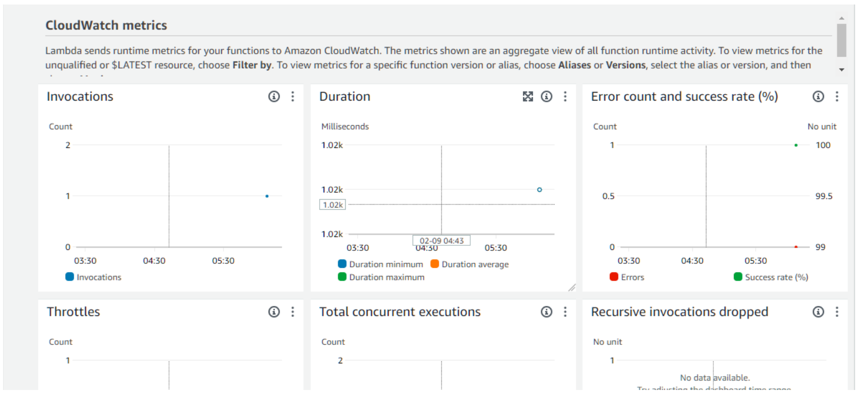
Monitoring

Metrics are numeric data measured at various time intervals (time series data) and service-level indicators (request rate, error rate, duration, CPU, etc.) The Lambda service automatically publishes a number of metrics for Lambda functions.

For monitoring and observing Lambda functions, the most important metrics are:

- **Errors:** whether errors are caused by logic or runtime errors in the code, or caused by interactions with the Lambda service or other services. These may also be caused by other factors such as lack of permissions or exceeding the assigned resources.
- **Execution time:** measuring average response times only provides a limited view of performance in distributed applications. It’s important to capture and monitor performance at percentile intervals (such as 95% and 99%) to measure the performance for the slowest 5% and 1% of requests.
- **Throttling:** serverless applications use scalable resources with Service Quotas to protect customers. Throttling may indicate that quotas are set incorrectly, there is an error in the application architecture, or traffic levels are beyond the expected limits.

All Lambda functions are automatically integrated with CloudWatch. Lambda automatically records a variety of standard metrics that are always published to CloudWatch metrics. If you navigate to a specific function in the Lambda
The **Monitor** tab's **Metrics** section in the Lambda console provides a quick view into integrated CloudWatch metrics for a single function:



CloudWatch metrics

Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose **Filter by**. To view metrics for a specific function version or alias, choose **Aliases** or **Versions**, select the alias or version, and then

Next, we will add a new feature to our web application using Amazon Rekognition.

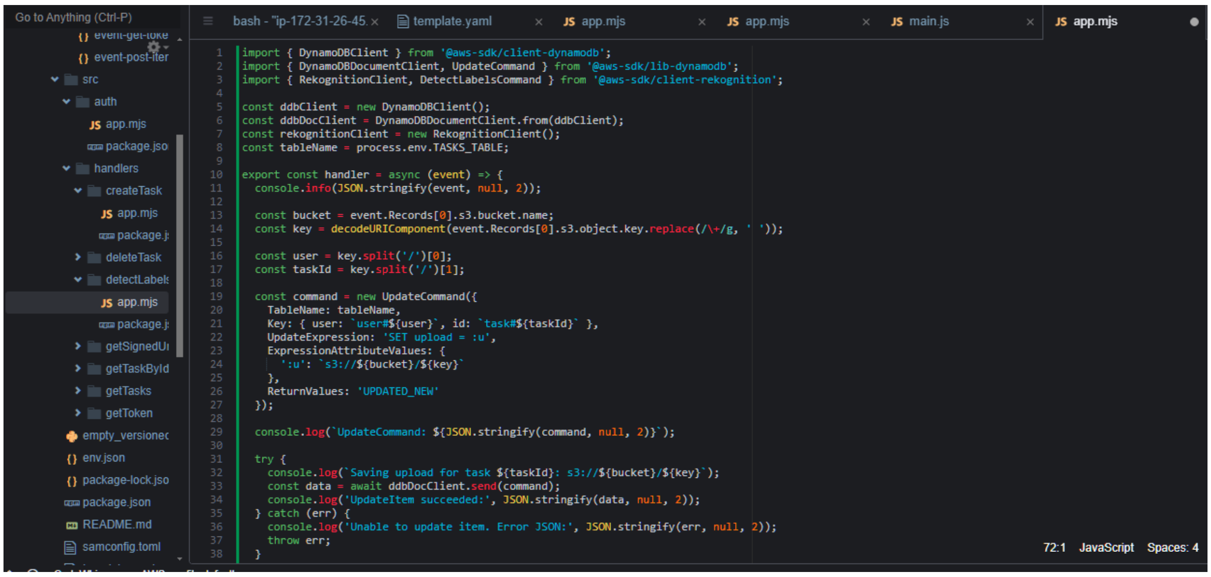
Step 5: Configure image metadata extraction: Amazon Rekognition

What is Amazon Rekognition?

Amazon Rekognition makes it easy to add image and video analysis to your applications using deep learning technology that requires no machine learning expertise to use. With Amazon Rekognition, you can identify objects, people, text, scenes, and activities in images and videos.
In this step, we will add a feature to detect objects in an uploaded image and apply labels to the detected objects.

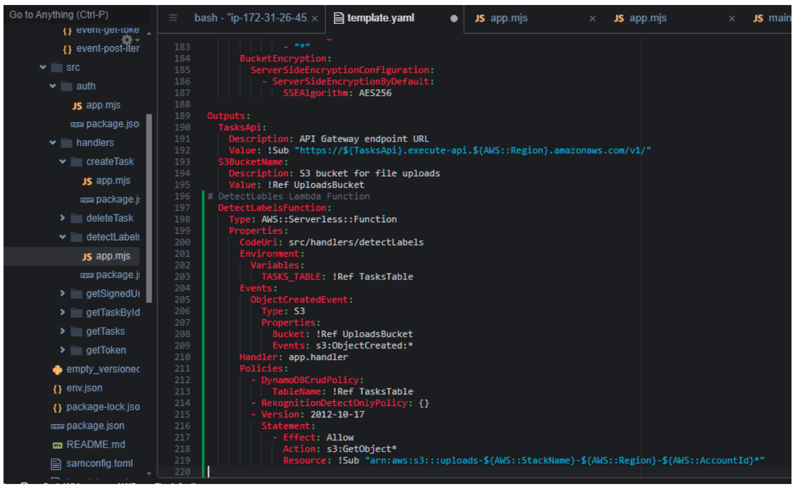
Configure Amazon Rekognition Integration

Edit the file named **sam/src/handlers/detectLabels/app.mjs** and paste the following code into it:



```javascript
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocumentClient, UpdateCommand } from '@aws-sdk/lib-dynamodb';
import { RekognitionClient, DetectLabelsCommand } from '@aws-sdk/client-rekognition';

const ddbClient = new DynamoDBClient();
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);
const rekognitionClient = new RekognitionClient();
const tableName = process.env.TASKS_TABLE;

export const handler = async (event) => {
  console.info(JSON.stringify(event, null, 2));

  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

  const user = key.split('/')[0];
  const taskId = key.split('/')[1];

  const command = new UpdateCommand({
    TableName: tableName,
    Key: { user: `user#${user}`, id: `task#${taskId}` },
    UpdateExpression: 'SET upload = :u',
    ExpressionAttributeValues: {
      ':u': `s3://${bucket}/${key}`
    },
    ReturnValues: 'UPDATED_NEW'
  });

  console.log(`UpdateCommand: ${JSON.stringify(command, null, 2)}`);

  try {
    console.log(`Saving upload for task ${taskId}: s3://${bucket}/${key}`);
    const data = await ddbDocClient.send(command);
    console.log('UpdateItem succeeded:', JSON.stringify(data, null, 2));
  } catch (err) {
    console.log('Unable to update item. Error JSON:', JSON.stringify(err, null, 2));
    throw err;
  }
```

Next we need to add a new **AWS::Serverless::Function** resource to **sam/template.yaml** under the **Resources** section. Remember to indent the code one level.



```yaml
              - "*"
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              SSEAlgorithm: AES256

Outputs:
  TasksApi:
    Description: API Gateway endpoint URL
    Value: !Sub "https://${TasksApi}.execute-api.${AWS::Region}.amazonaws.com/v1/"
  S3BucketName:
    Description: S3 bucket for file uploads
    Value: !Ref UploadsBucket
# DetectLables Lambda Function
  DetectLabelsFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: src/handlers/detectLabels
      Environment:
        Variables:
          TASKS_TABLE: !Ref TasksTable
      Events:
        ObjectCreatedEvent:
          Type: S3
          Properties:
            Bucket: !Ref UploadsBucket
            Events: s3:ObjectCreated:*
      Handler: app.handler
      Policies:
        - DynamoDBCrudPolicy:
            TableName: !Ref TasksTable
        - RekognitionDetectOnlyPolicy: {}
        - Version: 2012-10-17
          Statement:
            - Effect: Allow
              Action: s3:GetObject*
              Resource: !Sub "arn:aws:s3:::uploads-${AWS::StackName}-${AWS::Region}-${AWS::AccountId}*"
```

Notice the **ObjectCreatedEvent** property. This is the event that will trigger the Lambda function when an object is created in the uploads bucket.

Deploy the changes

With these modifications to the code, we can now build and deploy the changes to AWS. To do this, run the following commands from the **sam** directory:

cd ~/environment/serverless-tasks-webapp/sam

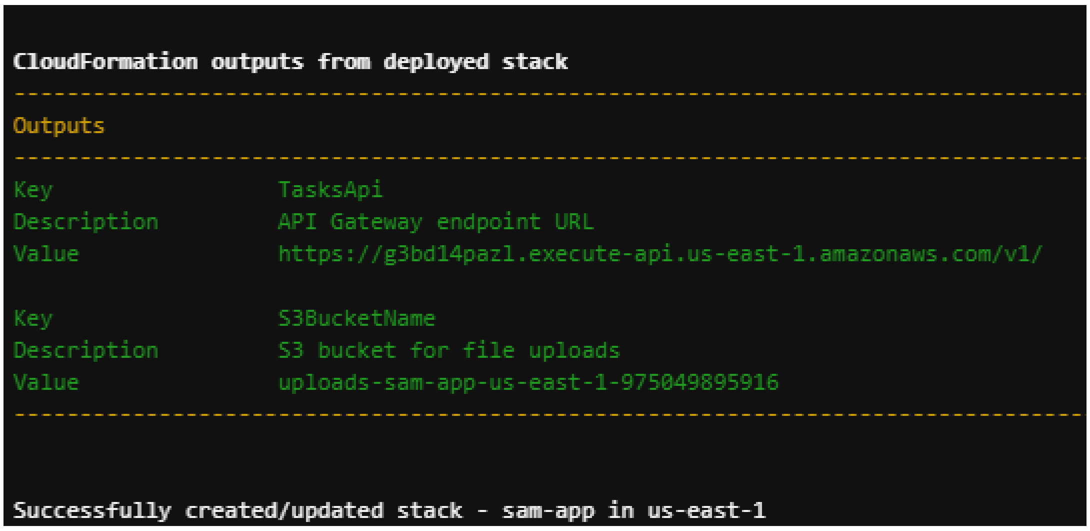
sam build

and then:

sam deploy –guided

sam deploy

you'll see the output as shown in below snap



```
CloudFormation outputs from deployed stack
---------------------------------------------------------------------------------
Outputs
---------------------------------------------------------------------------------
Key                 TasksApi
Description         API Gateway endpoint URL
Value               https://g3bd14pazl.execute-api.us-east-1.amazonaws.com/v1/

Key                 S3BucketName
Description         S3 bucket for file uploads
Value               uploads-sam-app-us-east-1-975049895916
---------------------------------------------------------------------------------


Successfully created/updated stack - sam-app in us-east-1
```

You are now ready to test Amazon Rekognition in your application.

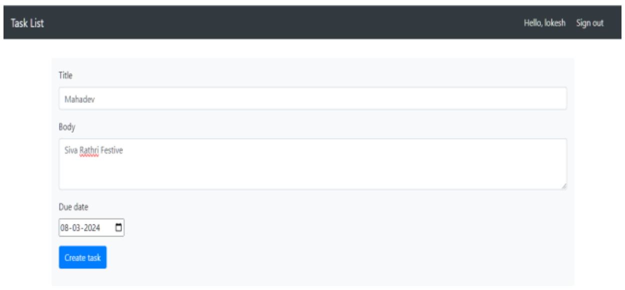
Upload an Image and Detect Labels

Label Detection in Amazon Rekognition

Amazon Rekognition Image can analyze images that are stored in an Amazon S3 bucket. Our web application allows you to upload an image as an attachment to task.
When the image is uploaded, it is stored in an S3 bucket. In the previous step, we configured a trigger to invoke a Lambda function when an object is created in the S3 bucket.
Try attaching an image to a task:

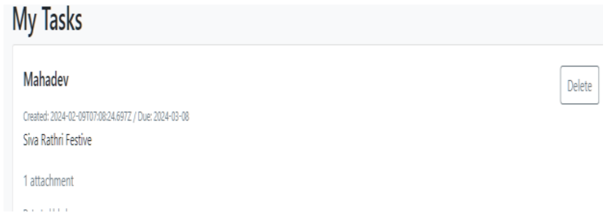
1. Click **Browse...**
2. Choose any JPEG format image file you like from your own computer, or from the sample images provided in the Github repository .
3. Click **Upload**

You will see a **Detecting Labels** progress message, and after a few seconds, you will see the detected labels in the task details:



Task List   Hello, lokesh   Sign out

Title
Mahadev
Body
Siva Rathri Festive
Due date
08-03-2024
Create task

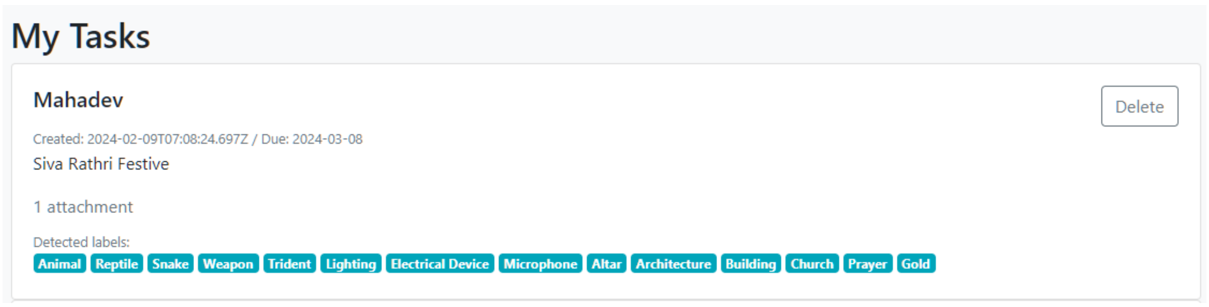


My Tasks

Mahadev   Delete

Created: 2024-02-09T07:08:24.697Z / Due: 2024-03-08
Siva Rathri Festive

1 attachment


My Tasks

Mahadev

Created: 2024-02-09T07:08:24.697Z / Due: 2024-03-08

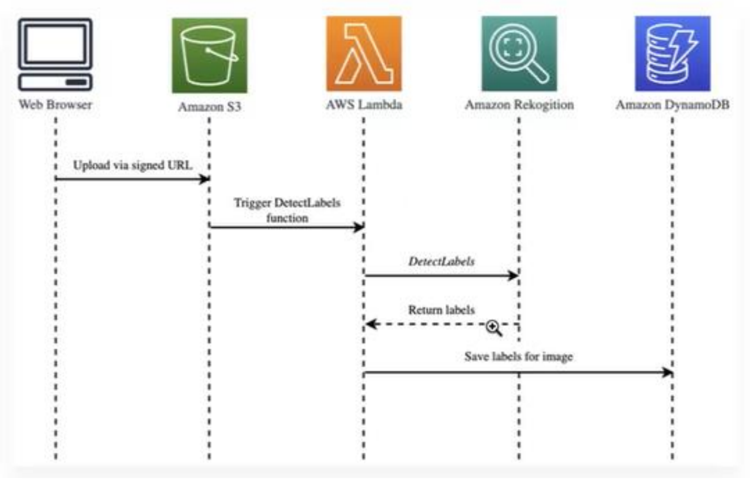
Siva Rathri Festive

1 attachment

Detected labels:

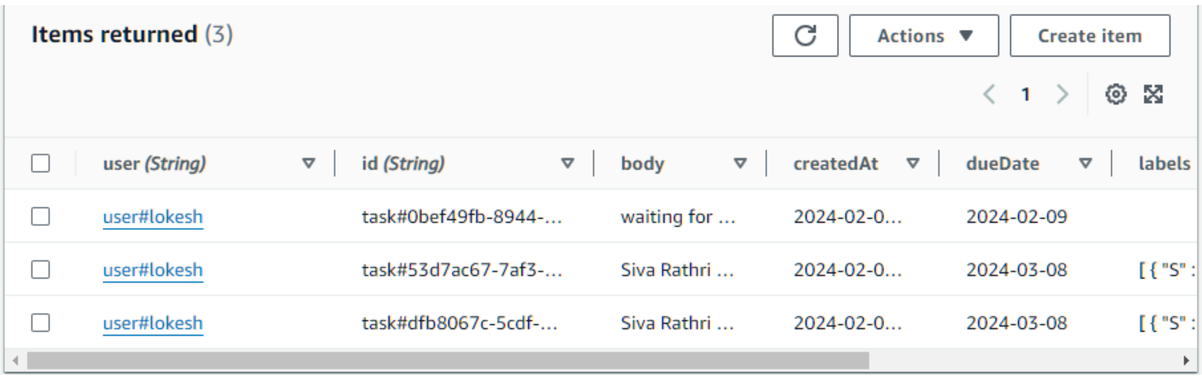
Animal Reptile Snake Weapon Trident Lighting Electrical Device Microphone Altar Architecture Building Church Prayer Gold

What just happened?



1. The image is uploaded to an S3 bucket using a presigned URL .
2. The S3 bucket is configured to trigger the **DetectLabels** Lambda function when an object is created in the bucket.
3. The Lambda function invokes the Amazon Rekognition **DetectLabels** function to detect labels in the image.
4. The detected labels are written to DynamoDB.
5. The web application refreshes the task details.

Notice how we added a major new feature to our web application without significantly changing the code. This demonstrates the power of serverless architectures — extensibility and modularity.


Items returned (3)

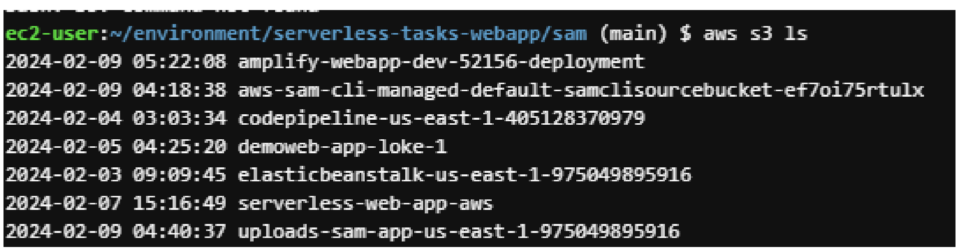
	user (String)	id (String)	body	createdAt	dueDate	labels
☐	user#lokesh	task#0bef49fb-8944-…	waiting for …	2024-02-0…	2024-02-09	
☐	user#lokesh	task#53d7ac67-7af3-…	Siva Rathri …	2024-02-0…	2024-03-08	[{ "S" :
☐	user#lokesh	task#dfb8067c-5cdf-…	Siva Rathri …	2024-02-0…	2024-03-08	[{ "S" :

Step 6: Terminate Resources

The simplest way to delete resources created by SAM is to use the **sam delete** command. However, before you do this, you need to ensure the S3 bucket for your image uploads is empty.

You can use the **aws s3 ls** command to see a list of your buckets. You should see an entry like this:

2021-09-13 21:28:43 uploads-tasks-app-us-east-1-123456789012



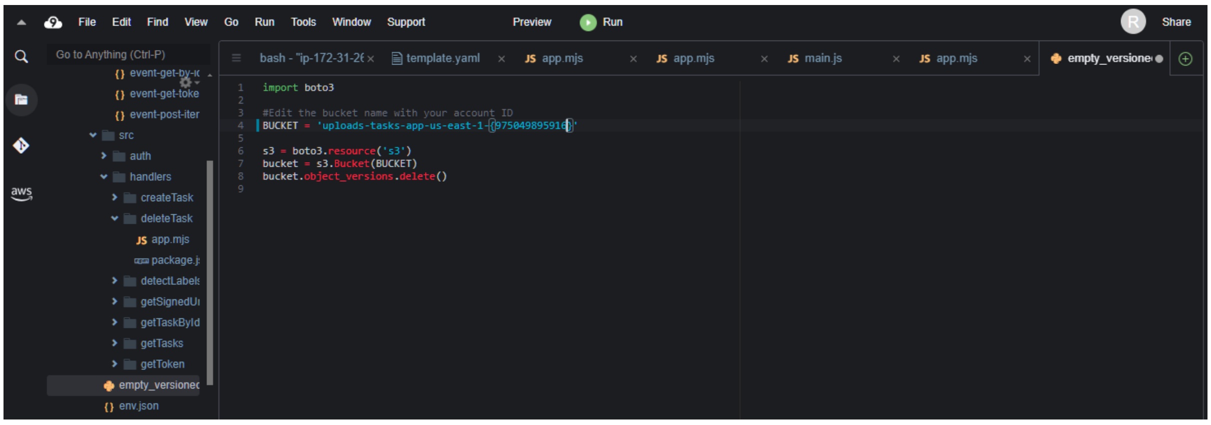
```
ec2-user:~/environment/serverless-tasks-webapp/sam (main) $ aws s3 ls
2024-02-09 05:22:08 amplify-webapp-dev-52156-deployment
2024-02-09 04:18:38 aws-sam-cli-managed-default-samclisourcebucket-ef7oi75rtulx
2024-02-04 03:03:34 codepipeline-us-east-1-405128370979
2024-02-05 04:25:20 demoweb-app-loke-1
2024-02-03 09:09:45 elasticbeanstalk-us-east-1-975049895916
2024-02-07 15:16:49 serverless-web-app-aws
2024-02-09 04:40:37 uploads-sam-app-us-east-1-975049895916
```

The bucket you're looking for is named like **uploads-tasks-app-us-east-1-123456789012** where **us-east-1** is your region, and the **123456789012** is your AWS account ID.

Go to your /**sam** directory

cd ~/environment/serverless-tasks-webapp/sam

The bucket for this workshop has versioning enabled and you must empty it through the AWS SDK. Locate the file **emtpy_versioned_bucket.py in ~/environment/serverless-tasks-webapp/sam** and modify it to add your AWS Account ID used from the output above. Save the file and then run the python file.

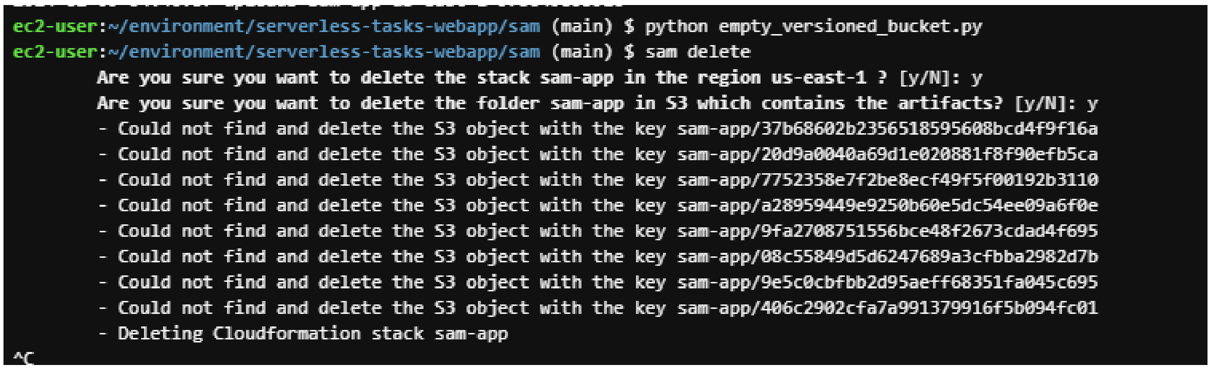


```python
import boto3

#Edit the bucket name with your account ID
BUCKET = 'uploads-tasks-app-us-east-1-{975049895916}'

s3 = boto3.resource('s3')
bucket = s3.Bucket(BUCKET)
bucket.object_versions.delete()
```

python empty_versioned_bucket.py

Now you may delete the resources created by SAM:

sam delete



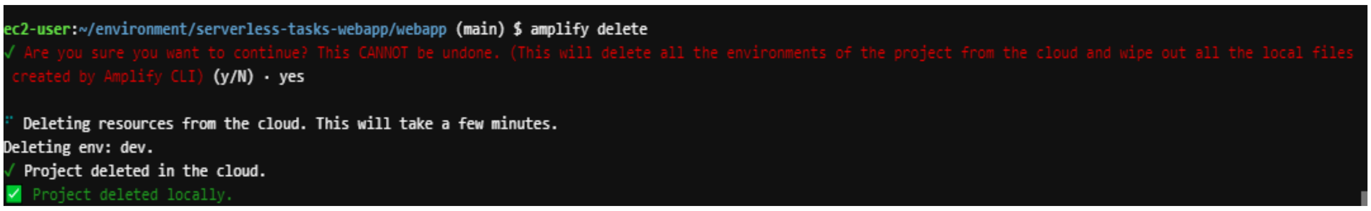
```
ec2-user:~/environment/serverless-tasks-webapp/sam (main) $ python empty_versioned_bucket.py
ec2-user:~/environment/serverless-tasks-webapp/sam (main) $ sam delete
        Are you sure you want to delete the stack sam-app in the region us-east-1 ? [y/N]: y
        Are you sure you want to delete the folder sam-app in S3 which contains the artifacts? [y/N]: y
        - Could not find and delete the S3 object with the key sam-app/37b68602b2356518595608bcd4f9f16a
        - Could not find and delete the S3 object with the key sam-app/20d9a0040a69d1e020881f8f90efb5ca
        - Could not find and delete the S3 object with the key sam-app/7752358e7f2be8ecf49f5f00192b3110
        - Could not find and delete the S3 object with the key sam-app/a28959449e9250b60e5dc54ee09a6f0e
        - Could not find and delete the S3 object with the key sam-app/9fa2708751556bce48f2673cdad4f695
        - Could not find and delete the S3 object with the key sam-app/08c55849d5d6247689a3cfbba2982d7b
        - Could not find and delete the S3 object with the key sam-app/9e5c0cbfbb2d95aeff68351fa045c695
        - Could not find and delete the S3 object with the key sam-app/406c2902cfa7a991379916f5b094fc01
        - Deleting Cloudformation stack sam-app
^C
```

Amplify Hosting

From the /**webapp** directory, you can delete the Amplify project by running this command:

cd ~/environment/serverless-tasks-webapp/webapp
amplify delete

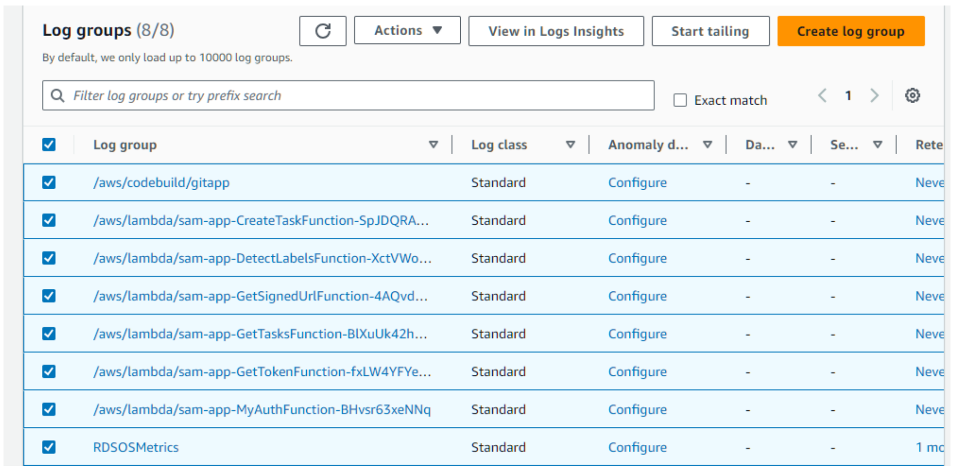
This command deletes all of the resources tied to the project from the cloud.



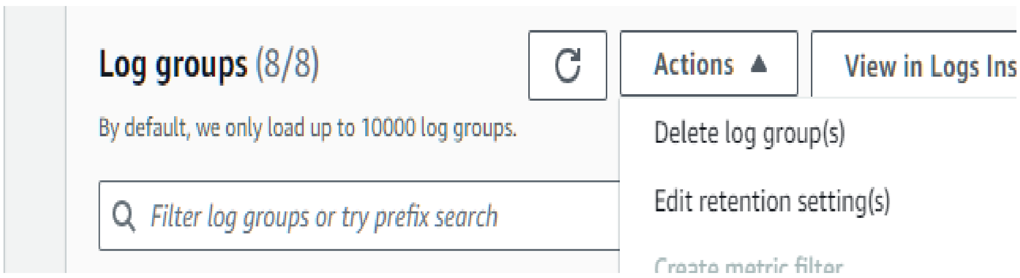
```
ec2-user:~/environment/serverless-tasks-webapp/webapp (main) $ amplify delete
✓ Are you sure you want to continue? This CANNOT be undone. (This will delete all the environments of the project from the cloud and wipe out all the local files
 created by Amplify CLI) (y/N) · yes

  Deleting resources from the cloud. This will take a few minutes.
Deleting env: dev.
✓ Project deleted in the cloud.
✅ Project deleted locally.
```

CloudWatch Logs

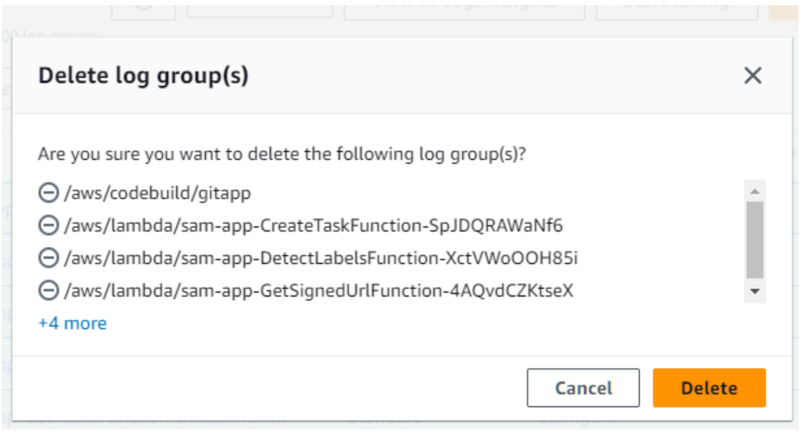
Additionally, you may want to delete any CloudWatch Logs log groups and streams created by the application.



☑	Log group	Log class	Anomaly d...	Da...	Se...	Rete
☑	/aws/codebuild/gitapp	Standard	Configure	-	-	Neve
☑	/aws/lambda/sam-app-CreateTaskFunction-SpJDQRA...	Standard	Configure	-	-	Neve
☑	/aws/lambda/sam-app-DetectLabelsFunction-XctVWo...	Standard	Configure	-	-	Neve
☑	/aws/lambda/sam-app-GetSignedUrlFunction-4AQvd...	Standard	Configure	-	-	Neve
☑	/aws/lambda/sam-app-GetTasksFunction-BlXuUk42h...	Standard	Configure	-	-	Neve
☑	/aws/lambda/sam-app-GetTokenFunction-fxLW4YFYe...	Standard	Configure	-	-	Neve
☑	/aws/lambda/sam-app-MyAuthFunction-BHvsr63xeNNq	Standard	Configure	-	-	Neve
☑	RDSOSMetrics	Standard	Configure	-	-	1 mo

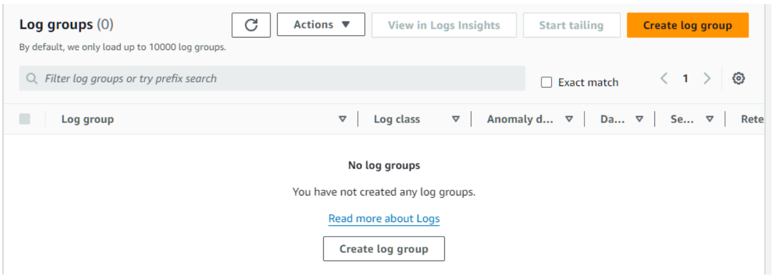


Log groups (8/8)
Actions ▲
View in Logs Ins
By default, we only load up to 10000 log groups.
Filter log groups or try prefix search
Delete log group(s)
Edit retention setting(s)
Create metric filter


Delete log group(s)

Are you sure you want to delete the following log group(s)?

/aws/codebuild/gitapp
/aws/lambda/sam-app-CreateTaskFunction-SpJDQRAWaNf6
/aws/lambda/sam-app-DetectLabelsFunction-XctVWoOOH85i
/aws/lambda/sam-app-GetSignedUrlFunction-4AQvdCZKtseX
+4 more


Log groups (0)

By default, we only load up to 10000 log groups.

No log groups

You have not created any log groups.

Conclusion

Conclusion

This workshop is just the beginning. Learn more about what you can build with serverless at Serverless Land .

About this workshop

Submitting feedback or suggestions for improvement

Code for the example applications and various templates is available on GitHub under MIT open source license. Please raise an issue or a pull request there for any changes related to the code.

Contributors to this workshop

- Mark Richman, Principal Solutions Architect
- Nati Goldberg, Sr. Solutions Architect

Documentation by
P. Lokesh (Mech)