**Session (AWS session)**

There are multiple way to create session for aws. Here few mentioned below:-

// Equivalent to session.New

sess, err := session.NewSessionWithOptions(session.Options{})

// Specify profile to load for the session's config

sess, err := session.NewSessionWithOptions(session.Options{

Profile: "profile\_name",

})

// Specify profile for config and region for requests

sess, err := session.NewSessionWithOptions(session.Options{

Config: aws.Config{Region: aws.String("us-east-2")},

Profile: "profile\_name",

})

// Force enable Shared Config support

sess, err := session.NewSessionWithOptions(session.Options{

SharedConfigState: SharedConfigEnable,

})

// Assume an IAM role with MFA prompting for token code on stdin

// \*\*\*THIS IS PREFERED TO USE\*\*\*

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: SharedConfigEnable,

}))

OR

// \*\*\*THIS IS PREFERED TO USE\*\*\*

sess, err := session.Must(session.NewSession(&aws.Config{ Region: aws.String("us-west-2")}, ))

​// NOTE:- session.New() is deprecated.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**DYNAMODB**

DynamoDB is a NOSQL data base offered by AWS, its similar to MongoDB

// Program to Create Table in DYNAMODB.

package main

import (

"fmt"

"log"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/awserr"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/dynamodb"

)

func main() {

// initilize aws session with valid credentials

sess, err := session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

// Config: aws.Config{Region: aws.String("us-east-1")},

})

if err != nil {

if aerr, ok := err.(awserr.Error); ok {

fmt.Println(aerr.Error())

}

// OR USE THIS

// aerr, ok := err.(awserr.Error)

// if (ok) {

// fmt.Println(aerr.Error())

// }

}

// create DynamoDB client

svc := dynamodb.New(sess)

//Specify Table Name to be created Movies

tableName := "Movies"

input := &dynamodb.CreateTableInput{

AttributeDefinitions: []\*dynamodb.AttributeDefinition{

{

AttributeName: aws.String("year"),

AttributeType: aws.String("N"),

},

{

AttributeName: aws.String("title"),

AttributeType: aws.String("S"),

},

},

KeySchema: []\*dynamodb.KeySchemaElement{

{

AttributeName: aws.String("year"),

KeyType: aws.String("HASH"), // partition key || primary key // REQUIRED

},

{

AttributeName: aws.String("title"),

KeyType: aws.String("RANGE"), // sort key

},

},

ProvisionedThroughput: &dynamodb.ProvisionedThroughput{

ReadCapacityUnits: aws.Int64(10),

WriteCapacityUnits: aws.Int64(10),

},

TableName: aws.String(tableName),

}

\_, err = svc.CreateTable(input)

if err != nil {

log.Fatalf("Got error calling CreateTable: %s", err)

}

fmt.Println("Created the table", tableName)

}

.....................................................................................................................................................

// Program to insert data Into DynamoDB

package main

import (

"fmt"

"log"

"strconv"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/dynamodb"

"github.com/aws/aws-sdk-go/service/dynamodb/dynamodbattribute"

)

type Item struct {

Year int `json:"year"`

Title string `json:"title"`

Plot string `json:"plot"`

Rating float64 `json:"rating"`

}

func main() {

// Initilize AWS session with valid Credentials.

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

}))

svc := dynamodb.New(sess)

// item to be Inserted in DynamoDB

item := Item{

Year: 2018,

Title: "Avengers",

Plot: "SCI-fci",

Rating: 4.0,

}

// Befor inserting into DynamoDB,

// convert Item struct into map[string]\*dynamodb.AttributeValue,

// because PutItemInput{} has field named as Item that accepts data of type map[string]\*dynamodb.AttributeValue,

// In all MarshalMap takes a struct and gives back a map of type map[string]\*dynamodb.AttributeValue,

av, err := dynamodbattribute.MarshalMap(item)

if err != nil {

log.Fatalf("Got error marshalling new movie item: %s", err)

}

// Table name

tableName := "Movies"

input := &dynamodb.PutItemInput{

Item: av, // this field requires data of type == map[string]\*dynamodb.AttributeValue

TableName: aws.String(tableName),

}

// execute PutItem method to insert data into DynamoDB

\_, err = svc.PutItem(input)

if err != nil {

log.Fatalf("Got error calling PutItem: %s", err)

}

year := strconv.Itoa(item.Year)

fmt.Println("Successfully added '" + item.Title + "' (" + year + ") to table " + tableName)

}

........................................................................................................................................................

​​// Program to GetItem/Fetch data from DynamoDB

package main

import (

"fmt"

"log"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/dynamodb"

"github.com/aws/aws-sdk-go/service/dynamodb/dynamodbattribute"

)

type Item struct {

Year int `json:"year"`

Title string `json:"title"`

}

// type data interface{}

func main() {

// Initilise session

sess := session.Must(session.NewSessionWithOptions(session.Options{

SharedConfigState: session.SharedConfigEnable,

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

}))

// create a DynamoDB service

svc := dynamodb.New(sess)

tableName := "Movies"

movieYear := "2020"

result, err := svc.GetItem(&dynamodb.GetItemInput{

TableName: aws.String(tableName),

Key: map[string]\*dynamodb.AttributeValue{

"year": {

N: aws.String(movieYear), // Key takes the values/primary key, which we use to find required data from DynamoDB

},

"title": {

S: aws.String("qwerty"),

},

},

})

if err != nil {

log.Fatalf("Got error calling GetItem: %s", err)

}

// if result.Item == nil {

// msg := "Could not find"

// return nil, errors.New(msg)

// }

// Output form result (FIELD Item) is of type == map[string]\*dynamodb.AttributeValue,

// with the below UnmarshalMap we converted this into type == Item

item := Item{}

err = dynamodbattribute.UnmarshalMap(result.Item, &item)

if err != nil {

panic(fmt.Sprintf("Failed to unmarshal Record, %v", err))

}

fmt.Println("Found item:")

fmt.Println("Year: ", item.Year)

fmt.Println("Title: ", item.Title)

}

.........................................................................................................................................

​​// Program to update data in DynamoDB

package main

import (

"fmt"

"log"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/dynamodb"

)

func main() {

// initilize session

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

}))

// Create service to access DynamoDB

svc := dynamodb.New(sess)

tableName := "Movies"

movieName := "Avengers"

movieYear := "2018"

movieRating := "0.5"

input := &dynamodb.UpdateItemInput{

// Takes values that we are about to Update

ExpressionAttributeValues: map[string]\*dynamodb.AttributeValue{

":r": {

N: aws.String(movieRating),

},

},

// Table Name where update is required

TableName: aws.String(tableName),

// To Identify/Find out which OBJECT needs update

Key: map[string]\*dynamodb.AttributeValue{

"year": {

N: aws.String(movieYear),

},

"title": {

S: aws.String(movieName),

},

},

// Its has Multiple options you can check.

ReturnValues: aws.String("UPDATED\_NEW"),

// Field Name which we are about to update

UpdateExpression: aws.String("set rating = :r"),

}

// execyte update function

\_, err := svc.UpdateItem(input)

if err != nil {

log.Fatalf("Got error calling UpdateItem: %s", err)

}

fmt.Println("Successfully updated '" + movieName + "' (" + movieYear + ") rating to " + movieRating)

}

...........................................................................................................................................................

// Program to delete data in DynamoDB

package main

import (

"fmt"

"log"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/dynamodb"

)

func main() {

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

}))

svc := dynamodb.New(sess)

tableName := "Movies"

movieName := "Avengers"

movieYear := "2018"

input := &dynamodb.DeleteItemInput{

Key: map[string]\*dynamodb.AttributeValue{

"year": {

N: aws.String(movieYear),

},

"title": {

S: aws.String(movieName),

},

},

TableName: aws.String(tableName),

}

\_, err := svc.DeleteItem(input)

if err != nil {

log.Fatalf("Got error calling DeleteItem: %s", err)

}

fmt.Println("Deleted '" + movieName + "' (" + movieYear + ") from table " + tableName)

}

....................................................................................................................................................

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

s3

// Program to Create a New Bucket

package main

import (

"fmt"

"os"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/s3"

)

func exitErrorf(msg string, args ...interface{}) {

fmt.Fprintf(os.Stderr, msg+"\n", args...)

os.Exit(1)

}

func main() {

// user need to pass bucket name form CLI in this program,

// you also can hard-code name here

if len(os.Args) != 2 {

exitErrorf("Bucket name missing!\nUsage: %s bucket\_name", os.Args[0])

}

bucket := os.Args[1]

sess := session.Must(session.NewSessionWithOptions(session.Options{

SharedConfigState: session.SharedConfigEnable,

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

}))

// Craete s3 service

svc := s3.New(sess)

\_, err := svc.CreateBucket(&s3.CreateBucketInput{

// This field takes bucket-name that we are about to create

Bucket: aws.String(bucket),

})

if err != nil {

exitErrorf("Unable to create bucket %q, %v", bucket, err)

}

// Wait until bucket is created before finishing

fmt.Printf("Waiting for bucket %q to be created...\n", bucket)

err = svc.WaitUntilBucketExists(&s3.HeadBucketInput{

Bucket: aws.String(bucket),

})

if err != nil {

exitErrorf("Error occurred while waiting for bucket to be created, %v", bucket)

}

fmt.Printf("Bucket %q successfully created\n", bucket)

}

................................................................................................................................................................

// Program to Upload file to s3

package main

import (

"fmt"

"os"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/s3/s3manager"

)

func exitErrorf(msg string, args ...interface{}) {

fmt.Fprintf(os.Stderr, msg+"\n", args...)

os.Exit(1)

}

func main() {

// User need to pass bucket name from CLI,

// OR you can hard-code here

if len(os.Args) != 3 {

exitErrorf("bucket and file name required\nUsage: %s bucket\_name filename", os.Args[0])

}

bucket := os.Args[1]

filename := os.Args[2]

file, err := os.Open(filename)

if err != nil {

exitErrorf("Unable to open file %q, %v", err)

}

defer file.Close()

// initilize session

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

}))

// Create s3manager service to upload file

uploader := s3manager.NewUploader(sess)

// s3manager service provide method to your file in AWS s3

\_, err = uploader.Upload(&s3manager.UploadInput{

Bucket: aws.String(bucket), // This field is Bucket Name

Key: aws.String(filename), // This field is your file-name that you want to UPLOAD

Body: file, // This field is of type == io.Reader, it takes readable content of the file, this can read from img or text files

})

if err != nil {

// Print the error and exit.

exitErrorf("Unable to upload %q to %q, %v", filename, bucket, err)

}

fmt.Printf("Successfully uploaded %q to %q\n", filename, bucket)

}

................................................................................................................................................................

// Progarm to Download file from s3 bucket.

package main

import (

"fmt"

"os"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/s3"

"github.com/aws/aws-sdk-go/service/s3/s3manager"

)

func exitErrorf(msg string, args ...interface{}) {

fmt.Fprintf(os.Stderr, msg+"\n", args...)

os.Exit(1)

}

func main() {

if len(os.Args) != 3 {

exitErrorf("Bucket and item names required\nUsage: %s bucket\_name item\_name", os.Args[0])

}

bucket := os.Args[1]

item := os.Args[2] // Object name to download

// This Creates file for you

file, err := os.Create(item)

if err != nil {

exitErrorf("Unable to open file %q, %v", item, err)

}

// initilize session

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

}))

// create s3manager service to download file from AWS s3

downloader := s3manager.NewDownloader(sess)

// this method downloads file from AWS and write it to file (io.File),

// returns number of bytes it has downloaded and error

numBytes, err := downloader.Download(file, &s3.GetObjectInput{

Bucket: aws.String(bucket), // bucket name from where file needs to download

Key: aws.String(item), // key is Just file name that need to be download.

})

if err != nil {

exitErrorf("Unable to download item %q, %v", item, err)

}

fmt.Println("Downloaded", file.Name(), numBytes, "bytes")

}

............................................................................................................................................................

// Program to delete file or Object from s3

package main

import (

"fmt"

"os"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/s3"

)

func exitErrorf(msg string, args ...interface{}) {

fmt.Fprintf(os.Stderr, msg+"\n", args...)

os.Exit(1)

}

func main() {

if len(os.Args) != 3 {

exitErrorf("Bucket and object name required\nUsage: %s bucket\_name object\_name",

os.Args[0])

}

bucket := os.Args[1]

obj := os.Args[2]

sess := session.Must(session.NewSessionWithOptions(session.Options{

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

SharedConfigState: session.SharedConfigEnable,

}))

svc := s3.New(sess)

\_, err := svc.DeleteObject(&s3.DeleteObjectInput{

Bucket: aws.String(bucket), // bucket Name

Key: aws.String(obj), // object Name or file Name to delete

})

if err != nil {

exitErrorf("Unable to delete object %q from bucket %q, %v", obj, bucket, err)

}

err = svc.WaitUntilObjectNotExists(&s3.HeadObjectInput{

Bucket: aws.String(bucket),

Key: aws.String(obj),

})

if err != nil {

exitErrorf("Error occurred while waiting for object %q to be deleted, %v", obj, err)

}

fmt.Printf("Object %q successfully deleted\n", obj)

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**API-GATEWAY**

SetUp and deployment instructions

Make sure you have installed serverless and dep:-

1. Serverless

Install:-

npm install -g serverless

2. aws-go-dep

Install:-

go get -u github.com/golang/dep/cmd/dep

steps :-

1. Create a Project folder

eg:-

api-gateway-demo

2. Open terminal at api-gateway-demo loaction:-

execute follwing commoands:-

mkdir src

cd src

serverless create -t aws-go-dep -p demo-services

cd ../

3. run command to open vscode:-

code .

4. enter command:-

connamd :- pwd

pwd, output you will get your ***current location*** like one below.

/home/opsadmin/api-gateway-demo

5. enter command:-

export GOPATH=”***current location***”

eg:-

export GOPATH=”/home/opsadmin/api-gateway-demo”

6. Check go path:-

command:- go env

7. run command

cd src/demo-services

8. run command:-

make

\*your setup Complete

refrence LINKS:-

SET-UP for go and serverless:-

https://www.serverless.com/blog/framework-example-golang-lambda-support

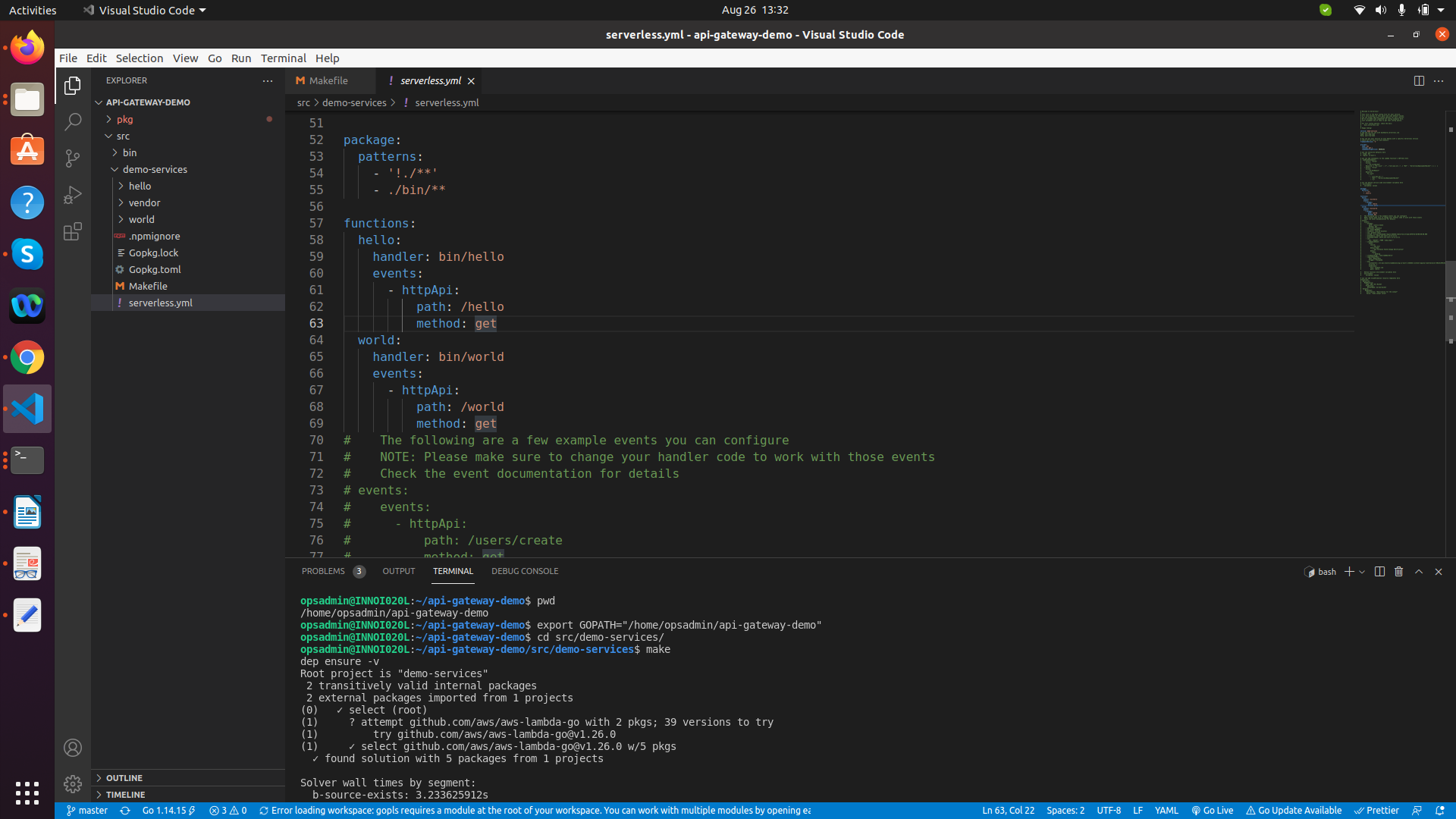
serverless and API-GATEWAY:-

[https://www.serverless.com/framework/docs/providers/aws/events/apigateway/#api-gateway](https://www.serverless.com/framework/docs/providers/aws/events/apigateway/" \l "api-gateway)

Note:-

Make sure your project is under [${GOPATH}/src](https://github.com/golang/go/wiki/GOPATH) directory

\*structure will look like this



------------------------------------------------------------------------------------------------------------------------

Now

Open serverless.yml

you will see:-

functions:

hello:

handler: bin/hello

events:

- httpApi:

path: /hello

method: get

here API-GATEWAY specification is like this.

**functions:**

**hello:** is your service

**handler:** is your binary file generated by make command, underlying command for make file is go build including ARCH and OS

**- httpApi or http:** is your configuration for API-GATEWAY

**path:** API access point path

**mehod:** request method type (GET, POST, PUT, PATCH, DELETE)

Write a lambda method service:-

package main

import (

"context"

"github.com/aws/aws-lambda-go/lambda"

"github.com/aws/aws-sdk-go/service/lambda"

)

func handler(ctx context.Context) string {

return "executed lambda successfully!!!"

}

func main() {

lambda.Start(handler)

}

Integrate above lambda method with **serverless** and **api-gateway**

-> open serverless.yml and write

example-service:

handler: bin/example\_service

events:

- httpApi:

path: /example-service

method: get

cors: true # setting cors = true means default cors configuration. which is also equivalent to example given below

---------OR------------

example-service:

handler: bin/example\_service

events:

- httpApi:

path: /example-service

method: get

cors: # this way is prefred to use, but try above for demo

origin: "\*"

headers:

- Content-Type

- X-Amz-Date

- Authorization

- X-Api-Key

- X-Amz-Security-Token

- X-Amz-User-Agent

\* your api-gateway integraetion with lambda and serverless is done

Now deploy:-

sls deploy --stage sandbox --region us-east-1 --account 938718425371

OR

sls deploy --stage sandbox --region us-east-1 --account 938718425371 --function function\_Name

**COGNITO**

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. Your users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, Google or Apple.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.

Type AdminCreateUserInput struct (Important FIELDS ) :-

// Specify "EMAIL" if email will be used to send the welcome message. Specify

// "SMS" if the phone number will be used. The default value is "SMS". More

// than one value can be specified.

DesiredDeliveryMediums []\*string `type:"list"`

// An array of name-value pairs that contain user attributes and attribute values

// to be set for the user to be created. You can create a user without specifying

// any attributes other than Username. However, any attributes that you specify

// as required (when creating a user pool or in the Attributes tab of the console)

// must be supplied either by you (in your call to AdminCreateUser) or by the

// user (when he or she signs up in response to your welcome message).

UserAttributes []\*AttributeType `type:"list"`

# Note:- The above type has other fields you can explore.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***CREATE USER WITH COGNITO***

package main

import (

"flag"

"fmt"

"os"

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/credentials/stscreds"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/cognitoidentityprovider"

)

func main() {

// create flags for emailID, userPool or userName

emailIDPtr := flag.String("e", "", "The email address of the user")

userPoolIDPtr := flag.String("p", "", "The ID of the user pool")

userNamePtr := flag.String("n", "", "The name of the user")

flag.Parse()

if \*emailIDPtr == "" || \*userPoolIDPtr == "" || \*userNamePtr == "" {

fmt.Println("You must supply an email address, user pool ID, and user name")

fmt.Println("Usage: go run CreateUser.go -e EMAIL-ADDRESS -p USER-POOL-ID -n USER-NAME")

os.Exit(1)

}

sess := session.Must(session.NewSessionWithOptions(session.Options{

SharedConfigState: session.SharedConfigEnable,

AssumeRoleTokenProvider: stscreds.StdinTokenProvider,

}))

svc := cognitoidentityprovider.New(sess)

newUserData := &cognitoidentityprovider.AdminCreateUserInput{

DesiredDeliveryMediums: []\*string{

aws.String("EMAIL"),

},

UserAttributes: []\*cognitoidentityprovider.AttributeType{

{

Name: aws.String("email"),

Value: aws.String(\*emailIDPtr),

},

},

UserPoolId: aws.String(\*userPoolIDPtr),

Username: aws.String(\*userNamePtr),

}

// Or you also use these functions

// newUserData.SetUserPoolId(\*userPoolIDPtr)

// newUserData.SetUsername(\*userNamePtr)

\_, err := svc.AdminCreateUser(newUserData)

if err != nil {

fmt.Println("Got error creating user:", err)

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CREATE-USER-POOL

package main

import (

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/cognitoidentityprovider"

"github.com/aws/aws-sdk-go/service/iam"

"fmt"

"os"

"strings"

)

func getRoleName(poolName string) string {

name := strings.Replace(poolName, "-", "", -1)

return name + "-SMS-Role"

}

// Creates Cognito user pool POOL\_NAME

//

// Usage:

// go run CognitoCreateUserPool.go POOL\_NAME

func main() {

if len(os.Args) < 2 {

fmt.Println("Pool name is required")

fmt.Println("Usage: go run", os.Args[0], "POOL\_NAME")

}

poolName := os.Args[2]

emailMsg := "{username} {####}" // Must match regex: [\p{L}\p{M}\p{S}\p{N}\p{P}\s\*]\*\{####\}[\p{L}\p{M}\p{S}\p{N}\p{P}\s\*]\*

emailSubject := "AWS TW Chat"

smsMsg := "{username} {####}" // Must match regex: .\*\{####\}.\*

waitDays := int64(1)

emailVerifyMsg := "{####}" // Must match regex: [\p{L}\p{M}\p{S}\p{N}\p{P}\s\*]\*\{####\}[\p{L}\p{M}\p{S}\p{N}\p{P}\s\*]\*

emailVerifySub := "AWS TW Chat"

smsAuthMsg := "{####}" // Must match regex: .\*\{####\}.\*

smsVerifyMsg := "{####}" // Must match regex: .\*\{####\}.\*

// Initialize a session that the SDK will use to load configuration,

// credentials, and region from the shared config file. (~/.aws/config).

sess := session.Must(session.NewSessionWithOptions(session.Options{

SharedConfigState: session.SharedConfigEnable,

}))

// // Create SMS role so pool can msg new users on your behalf

iamSvc := iam.New(sess)

doc := "{ \"Version\": \"2012-10-17\", \"Statement\": [ { \"Sid\": \"\", \"Effect\": \"Allow\", \"Principal\": { \"Service\": \"cognito-idp.amazonaws.com\" }, \"Action\": \"sts:AssumeRole\" } ] }"

// Create SMS role with pool name, less any hyphens

roleName := getRoleName(poolName) // Required

path := "/service-role/"

iamResp, iamErr := iamSvc.CreateRole(

&iam.CreateRoleInput{

AssumeRolePolicyDocument: &doc,

RoleName: &roleName,

Path: &path})

if iamErr != nil {

fmt.Println("Could not create role")

os.Exit(1)

}

roleArn := iamResp.Role.Arn

roleID := iamResp.Role.RoleId

// Create Cognito client

cgSvc := cognitoidentityprovider.New(sess)

params := &cognitoidentityprovider.CreateUserPoolInput{

PoolName: &poolName, // Required

AdminCreateUserConfig: &cognitoidentityprovider.AdminCreateUserConfigType{

AllowAdminCreateUserOnly: aws.Bool(false), // false == users can sign themselves up

InviteMessageTemplate: &cognitoidentityprovider.MessageTemplateType{

EmailMessage: &emailMsg, // Welcome message to new users

EmailSubject: &emailSubject, // Welcome subject to new users

SMSMessage: &smsMsg,

},

UnusedAccountValidityDays: &waitDays, // How many days to wait before rescinding offer

},

AutoVerifiedAttributes: []\*string{ // Auto-verified means the user confirmed the SNS message

aws.String("email"), // Required; either email or phone\_number

aws.String("phone\_number"),

},

EmailVerificationMessage: &emailVerifyMsg,

EmailVerificationSubject: &emailVerifySub,

Policies: &cognitoidentityprovider.UserPoolPolicyType{

PasswordPolicy: &cognitoidentityprovider.PasswordPolicyType{

MinimumLength: aws.Int64(6), // Require a password of at least 6 chars

RequireLowercase: aws.Bool(false),

RequireNumbers: aws.Bool(false),

RequireSymbols: aws.Bool(false),

RequireUppercase: aws.Bool(false),

},

},

Schema: []\*cognitoidentityprovider.SchemaAttributeType{

{ // Required

AttributeDataType: aws.String("String"),

DeveloperOnlyAttribute: aws.Bool(false),

Mutable: aws.Bool(false),

Name: aws.String("user\_name"),

Required: aws.Bool(false),

StringAttributeConstraints: &cognitoidentityprovider.StringAttributeConstraintsType{

MaxLength: aws.String("64"), // user name can be up to 64 chars

MinLength: aws.String("3"), // or as few as 3 chars

},

},

},

SmsAuthenticationMessage: &smsAuthMsg,

SmsConfiguration: &cognitoidentityprovider.SmsConfigurationType{

SnsCallerArn: roleArn, // Required

ExternalId: roleID,

},

SmsVerificationMessage: &smsVerifyMsg,

}

fmt.Println("")

cgResp, cgErr := cgSvc.CreateUserPool(params)

if cgErr != nil {

fmt.Println("Could not create user pool")

os.Exit(1)

}

fmt.Println("")

fmt.Println(cgResp)

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LISTUSERPOOL

package main

import (

"github.com/aws/aws-sdk-go/aws"

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/cognitoidentityprovider"

"fmt"

"os"

)

func main() {

// Initialize a session that the SDK will use to load configuration,

// credentials, and region from the shared config file. (~/.aws/config).

sess := session.Must(session.NewSessionWithOptions(session.Options{

SharedConfigState: session.SharedConfigEnable,

}))

// Create Cognito client

svc := cognitoidentityprovider.New(sess)

max := int64(10)

result, err := svc.ListUserPools(

&cognitoidentityprovider.ListUserPoolsInput{

MaxResults: &max,

}) // .ListBuckets(nil)

if err != nil {

fmt.Println("Could not list user pools")

os.Exit(1)

}

fmt.Println("User pools:")

fmt.Println("")

for \_, pool := range result.UserPools {

fmt.Println("Name: " + aws.StringValue(pool.Name))

fmt.Println("ID: " + aws.StringValue(pool.Id))

fmt.Println("Created: " + aws.TimeValue(pool.CreationDate).String())

fmt.Println("")

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LISTUSERS

package main

import (

"github.com/aws/aws-sdk-go/aws/session"

"github.com/aws/aws-sdk-go/service/cognitoidentityprovider"

"flag"

"fmt"

"os"

)

func main() {

userPoolIDPtr := flag.String("p", "", "The ID of the user pool")

flag.Parse()

if \*userPoolIDPtr == "" {

fmt.Println("You must supply a user pool ID")

fmt.Println("Usage: go run CreateUser.go -p USER-POOL-ID")

os.Exit(1)

}

// Initialize a session that the SDK will use to load

// credentials from the shared credentials file ~/.aws/credentials.

sess := session.Must(session.NewSessionWithOptions(session.Options{

SharedConfigState: session.SharedConfigEnable,

}))

cognitoClient := cognitoidentityprovider.New(sess)

results, err := cognitoClient.ListUsers(

&cognitoidentityprovider.ListUsersInput{

UserPoolId: userPoolIDPtr})

if err != nil {

fmt.Println("Got error listing users")

os.Exit(1)

}

// Show their names an email addresses

for \_, user := range results.Users {

attributes := user.Attributes

for \_, a := range attributes {

if \*a.Name == "name" {

fmt.Println("Name: " + \*a.Value)

} else if \*a.Name == "email" {

fmt.Println("Email: " + \*a.Value)

}

}

fmt.Println("")

}

}

IMPORTANT LINKS TO UNDERSTAND MORE:-

https://docs.aws.amazon.com/code-samples/latest/catalog/go-cognito-CognitoCreateUser.go.html

https://docs.aws.amazon.com/code-samples/latest/catalog/go-cognito-CognitoCreateUserPool.go.html

https://docs.aws.amazon.com/code-samples/latest/catalog/go-cognito-CognitoListUserPools.go.html

https://docs.aws.amazon.com/code-samples/latest/catalog/go-cognito-CognitoListUsers.go.html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SNS**