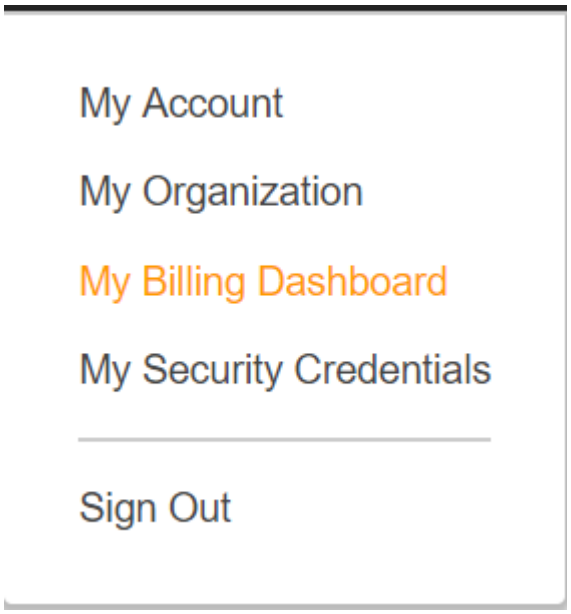
Run Go codes with AWS Lambda under AWS CLI in Win10

1. AWS CLI

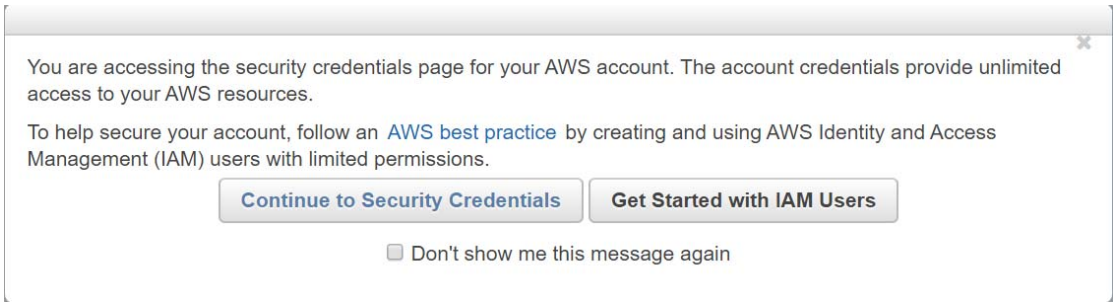
- Use root user (Not recommended)

This user has the highest access rights, so it is not safe. We'd better create IAM admin user.

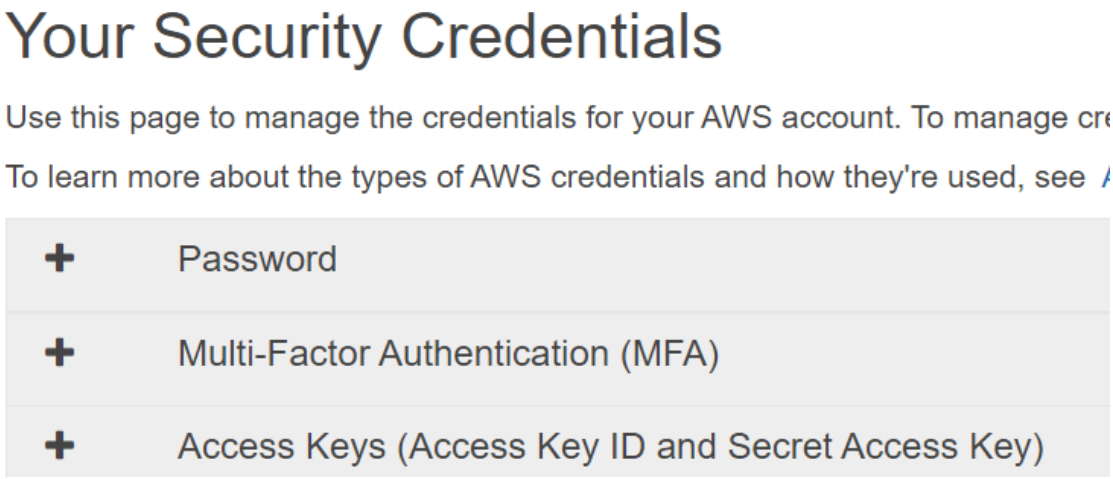
1) Log in AWS Console, click on username and select My Security Credentials.



2) Continue to Security Credentials



3) Create new access key under Access Keys (Access Key ID and Secret Access Key)



4) In command line, use 'aws configure' to configure the access keys.

- Admin user

1) The steps are similar to ones when creating user with PowerUserAccess. User with PowerUserAccess has no right to use the IAM roles.
2) Use AdministratorAccess permission to create admin user. User with AdministratorAccess has the right to use the IAM roles.
3) Log in the new admin user to check the rights in AWS Console.

2. Run Go code with AWS Lambda under AWS CLI on Win10

Reference:

https://github.com/eawsy/aws-lambda-go

https://github.com/eawsy/aws-lambda-go-shim

- In command line, execute go get -u -d github.com/eawsy/aws-lambda-go/...

Eawsy will be downloaded to the Go working path, such as C:\Users\Joycelong\go\src.

- Save the sample code in a file named handler.go, such as in C:\Users\Joycelong\go

1) The handler.go file is where resides the main entrypoint of our AWS Lambda function. There is no restriction on how many files and dependencies we can have, nor on how we must name our files and functions. Nevertheless however eawsy/aws-lambda-go advocate to retain the handler name and to use Handle as the name of our entrypoint.

For a seamless experience eawsy/aws-lambda-go leverage Go 1.8 plugins to separate the shim from our code. At run time, AWS Lambda loads pre-compiled shim which in turn loads our code (our plugin). A plugin is a Go main package and that is why our entrypoint must be in the main package. This restriction only applies to our entrypoint and we are free to organize the rest of our code in different packages.

2) The handler follows the AWS Lambda programming model by:

Taking 2 parameters:

Event – Event data is automatically json unmarshalled and passed as the first argument.

Also support json.RawMessage type, any other valid Go type or even our own custom type.

Context – Context information is passed as the second argument as explained above.

Returning 2 values:

Result – The first return value is automatically json marshalled and forwarded back to the client.

Support the well known interface{} type, any other valid Go type or even our own custom type to leverage fine grained json marshalling.

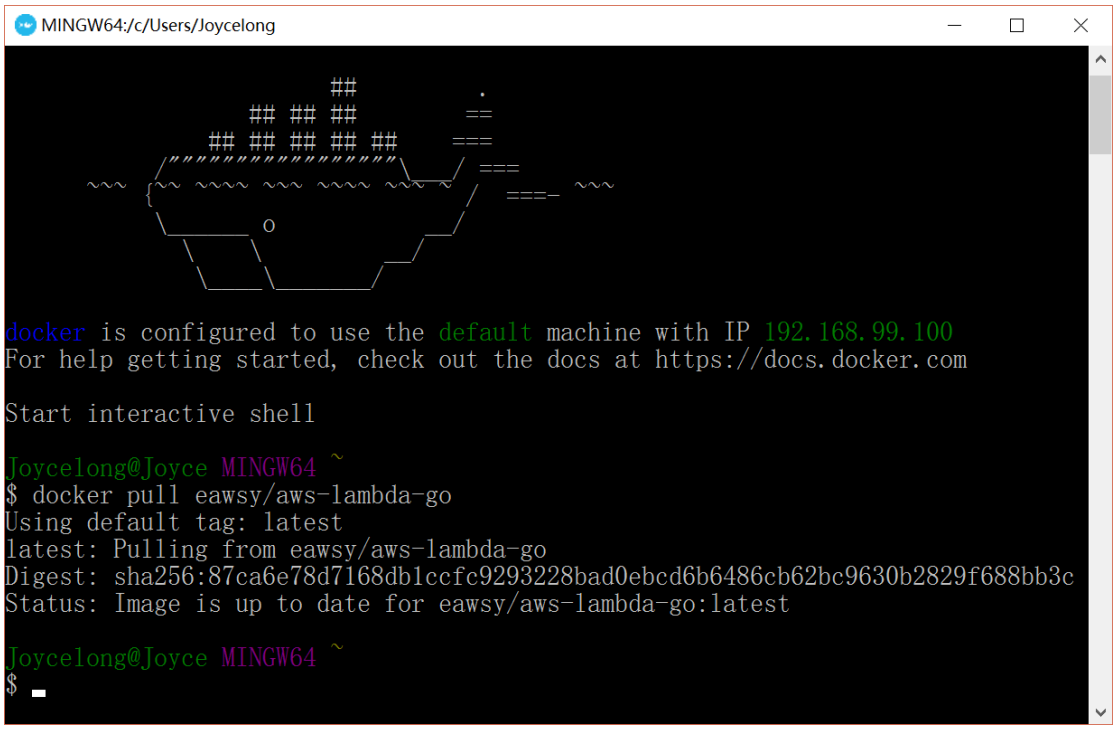
Error – The second return value notifies AWS Lambda an error occurred during execution. As expected it prevails over the first return value.

In the course of a normal and controlled execution flow, we can notify AWS Lambda an error occurred by returning an error.

```
package main

import (
```

```
    "encoding/json"

    "github.com/eawsy/aws-lambda-go/service/lambda/runtime"
)

func handle(evt json.RawMessage, ctx *runtime.Context) (interface{}, error) {
    return "Hello, World!", nil
}

func init() {
    runtime.HandleFunc(handle)
}

func main() {}
```

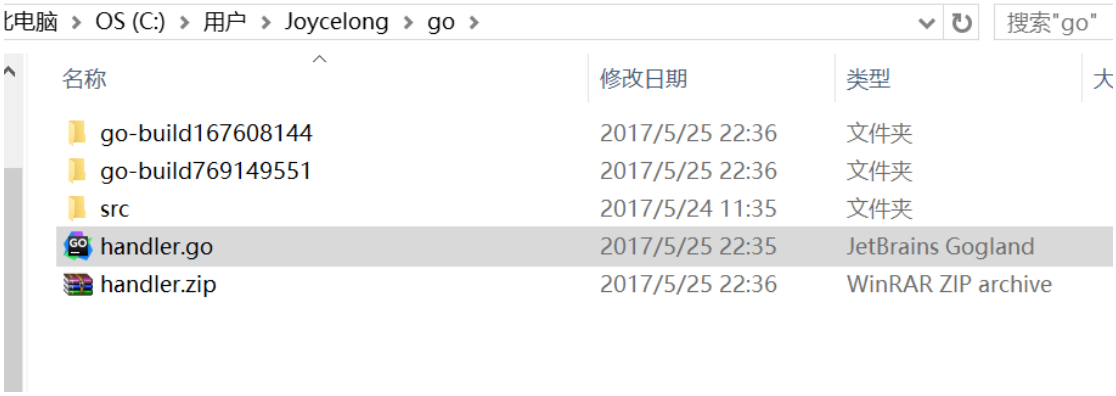
- Run Docker Quickstart Terminal (terminal) to start Docker
- In terminal, execute docker pull eawsy/aws-lambda-go



- In Win10, need change the sample code a little bit for the import part

./src/github.com/eawsy/aws-lambda-go/service/lambda/runtime

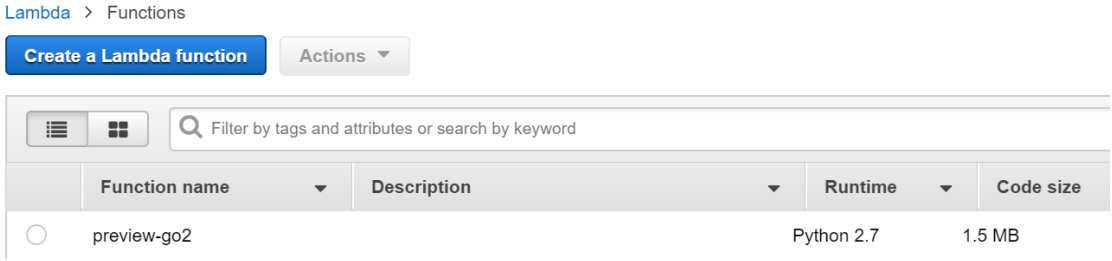
- In terminal, enter the path with sample code handler.go and execute docker run --rm -v c:/go -v $PWD:/tmp eawsy/aws-lambda-go



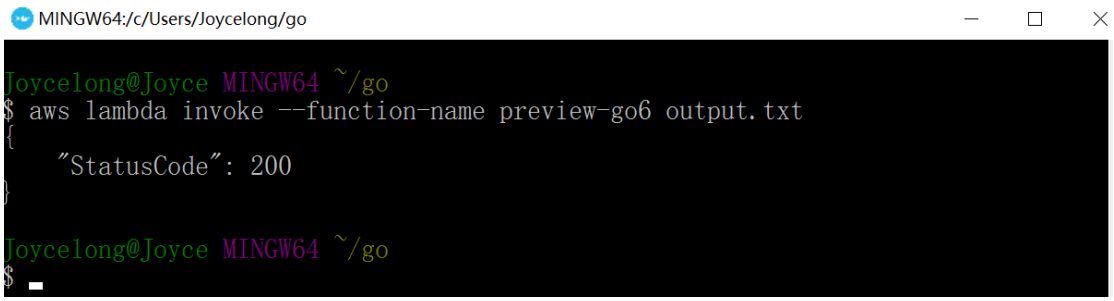
If a new file – handler.zip is successfully generated after about 1 minute, the execution is correct.

- In terminal, execute aws lambda create-function --function-name preview-go2 --runtime python2.7 --handler handler.handle --zip-file fileb://handler.zip --role arn:aws:iam::663833079xxx:role/AWSPowerUser

- Check AWS console, the new function is created successfully.



- In terminal, execute aws lambda invoke --function-name preview-go2 output.txt



Open the log file output.txt to check the result

- We can also execute the function in AWS console. Click Test to run the function and check the result