School of Computing

Creating GUIs for Lambda function POST handlers

BT3103 - Week 9
2019/2020 Semester 1

Associate Professor Chris BOESCH

The Challenges

1. How do we host a graphical user interface (GUI) for our lambda function that is able to POST back to our lambda function?
2. How do we make it as easy and error-free as possible for all of our collaborators to deploy and test their own copy of the latest code on Github?

A few ways to meet these challenges

1. **Single file** - Add a GET handler in your lambda function to return some html.
2. **Two files** - Add a GET handler in your lambda function to the contents of a locally saved index.html in the lambda file system.
3. **Proxied API** - Configure the API with a proxy so that you can return different files (index.html, main.js, app.css, etc.) on GETs.
4. **Github Pages** - Host the GUI from Github using Github pages.
5. **Sam Template** - Add a SAM template to automate the configuration, packaging, and deployment of all files.
6. **Github Action** - Configure a Github action to redeploy the latest code on commits to a repository.

Single File

Keep all of the GUI code (html, css, javascript) in a separate index.html file and leave all other code in the lambda handler file (lambday_handler.py) (example)

Pros:

- Collaborators can simply create a new lambda function and copy the latest code for each of the two files on on Github into the lambda editor and save their new function.

Cons:

- All of the python/javascript POST-handling code and GUI html, javascript, and css code will be in a single file. This could be difficult to understand, edit, test, and debug.
- The code editor will only be able to highlight and format the code as a single language.

Demo - minimal code single file version

Demo: https://o25djybnr2.execute-api.us-east-1.amazonaws.com/default/minimalFiveTabActivity

Repo: https://github.com/Ourstress/minimalCodeLambdaFunction/blob/master/singleFileFiveTabActivity.py

Demo

The demo lambda function consists of 2 parts:
1) frontend Vue app

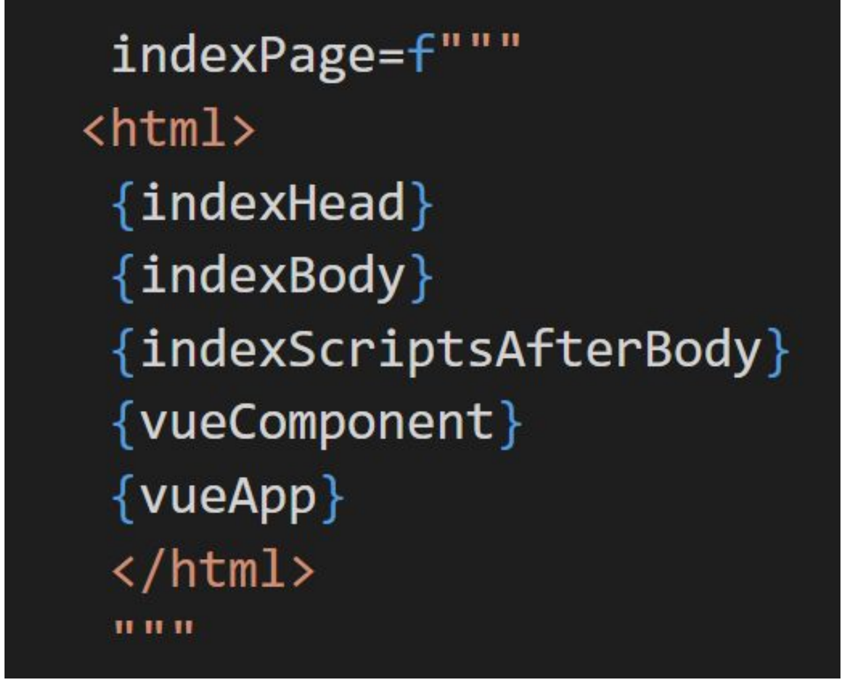
2) lambda function code

Demo

Front end vue app part

Notes:

- indexPage contains the vue app
- Within the lambda function, index page is just a multiline string
- We use f-string to concatenate the various parts of the vue app together



```
 indexPage=f"""
<html>
 {indexHead}
 {indexBody}
 {indexScriptsAfterBody}
 {vueComponent}
 {vueApp}
 </html>
 """
```

Demo

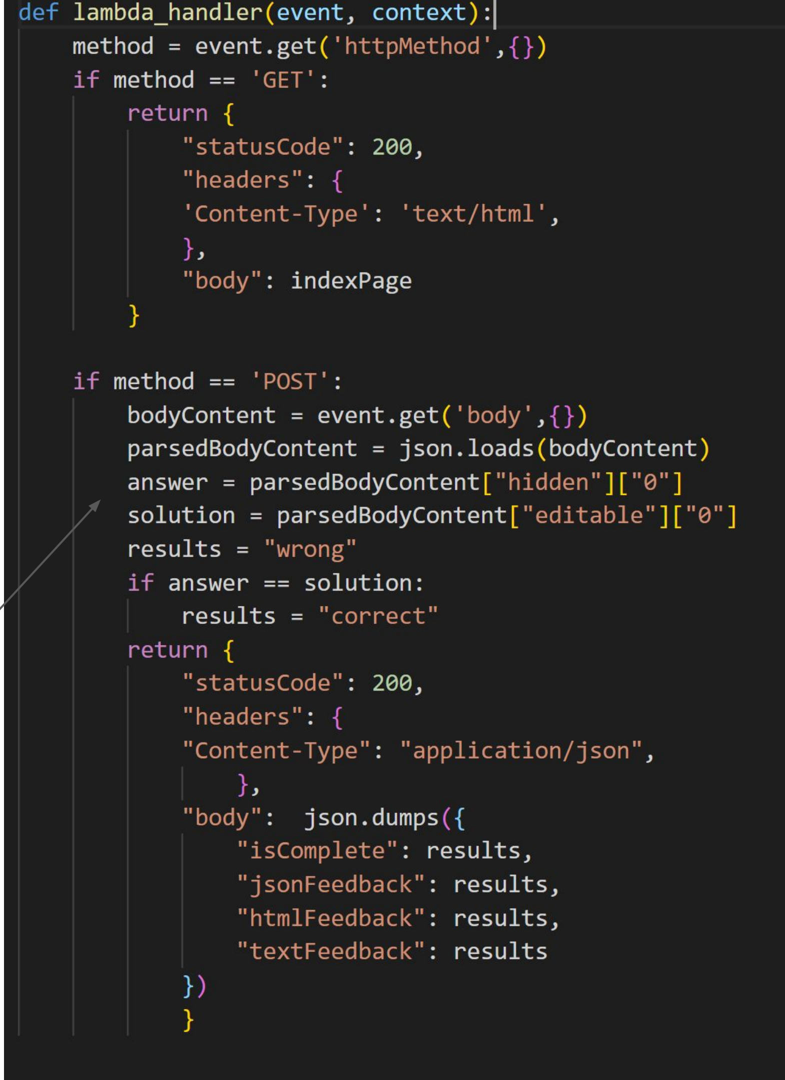
Lambda function code part

GET:

On http GET request, we return indexPage to the user's browser

POST:

When the user clicks on "submit" in the Vue app, the vue app sends some content to our lambda function api

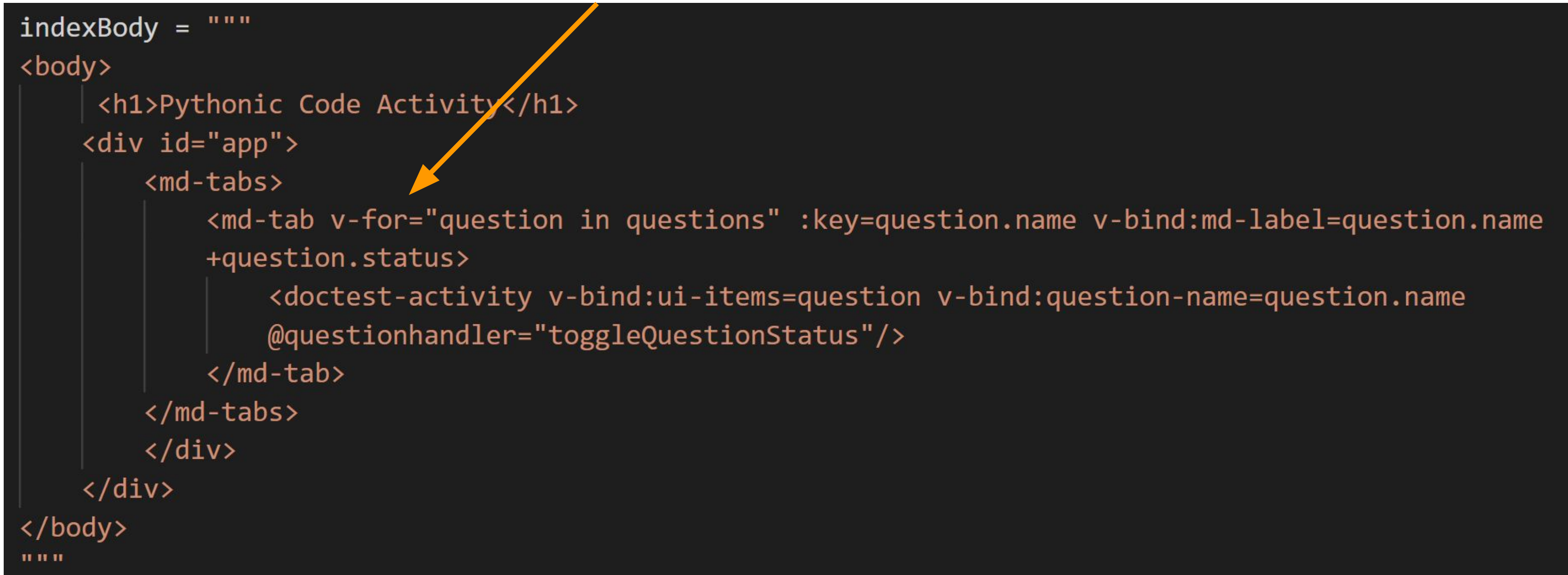


```python
def lambda_handler(event, context):
    method = event.get('httpMethod',{})
    if method == 'GET':
        return {
            "statusCode": 200,
            "headers": {
            'Content-Type': 'text/html',
            },
            "body": indexPage
        }

    if method == 'POST':
        bodyContent = event.get('body',{})
        parsedBodyContent = json.loads(bodyContent)
        answer = parsedBodyContent["hidden"]["0"]
        solution = parsedBodyContent["editable"]["0"]
        results = "wrong"
        if answer == solution:
            results = "correct"
        return {
            "statusCode": 200,
            "headers": {
            "Content-Type": "application/json",
                },
            "body":  json.dumps({
                "isComplete": results,
                "jsonFeedback": results,
                "htmlFeedback": results,
                "textFeedback": results
            })
            }
```

Demo

Basically, the demo app consists of one Vue app and one Vue component

The component is called doctest-activity.

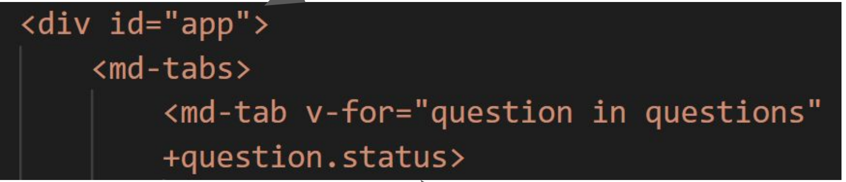


```
indexBody = """
<body>
     <h1>Pythonic Code Activity</h1>
    <div id="app">
        <md-tabs>
            <md-tab v-for="question in questions" :key=question.name v-bind:md-label=question.name
            +question.status>
                <doctest-activity v-bind:ui-items=question v-bind:question-name=question.name
                @questionhandler="toggleQuestionStatus"/>
            </md-tab>
        </md-tabs>
        </div>
    </div>
</body>
"""
```

Demo

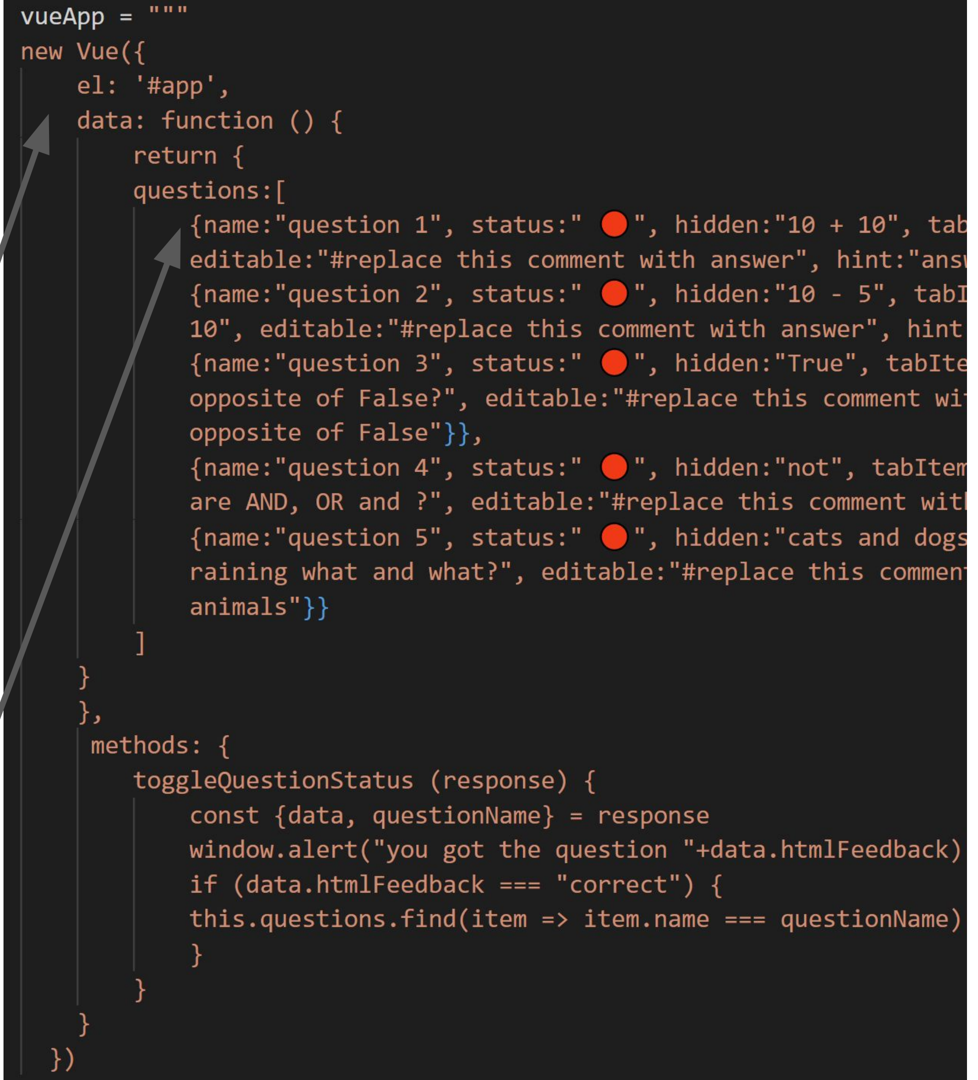
The vue app basically consists of data and a toggleQuestionStatus method.

Notice 1) the #app id at the top most div - inside of this div is the vue app



```
<div id="app">
    <md-tabs>
        <md-tab v-for="question in questions"
        +question.status>
```

Notice 2) we loop over the items in questions one by one. Each question gets an md-tab

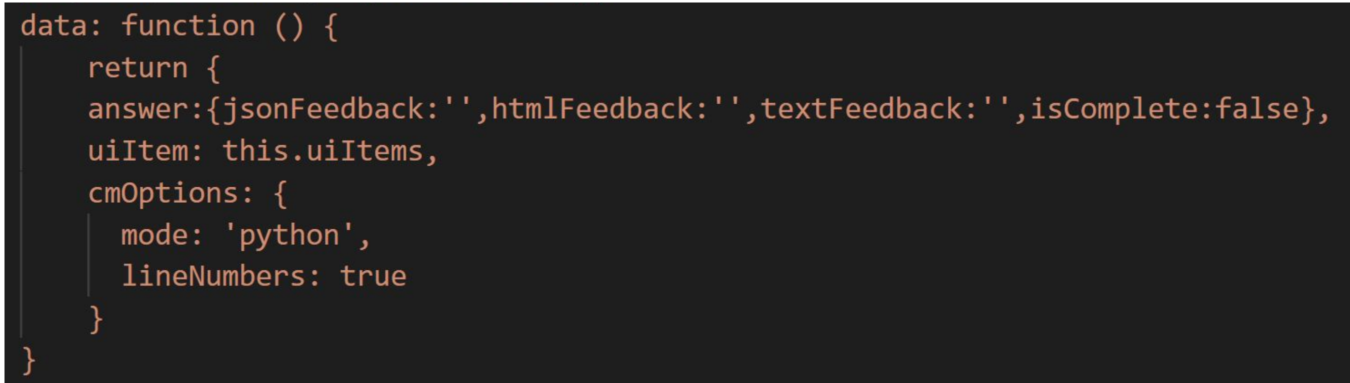


```
vueApp = """
new Vue({
    el: '#app',
    data: function () {
        return {
        questions:[
            {name:"question 1", status:"🔴", hidden:"10 + 10", tab
            editable:"#replace this comment with answer", hint:"ans
            {name:"question 2", status:"🔴", hidden:"10 - 5", tabI
            10", editable:"#replace this comment with answer", hint
            {name:"question 3", status:"🔴", hidden:"True", tabIte
            opposite of False?", editable:"#replace this comment wi
            opposite of False"}},
            {name:"question 4", status:"🔴", hidden:"not", tabItem
            are AND, OR and ?", editable:"#replace this comment wit
            {name:"question 5", status:"🔴", hidden:"cats and dogs
            raining what and what?", editable:"#replace this commen
            animals"}}
        ]
    }
    },
     methods: {
        toggleQuestionStatus (response) {
            const {data, questionName} = response
            window.alert("you got the question "+data.htmlFeedback)
            if (data.htmlFeedback === "correct") {
            this.questions.find(item => item.name === questionName)
            }
        }
    }
 })
```

Demo

Vue component called doctest-activity is made up of four parts

1) Props - it gets props from the vue app questions
2) Data - the vue component has stores data from the post response in ***answer***, props it received in ***uiltem*** and information to pass to codemirror in ***cmOptions***
3) Method - to post back to the lambda function API
4) Template - to display the component

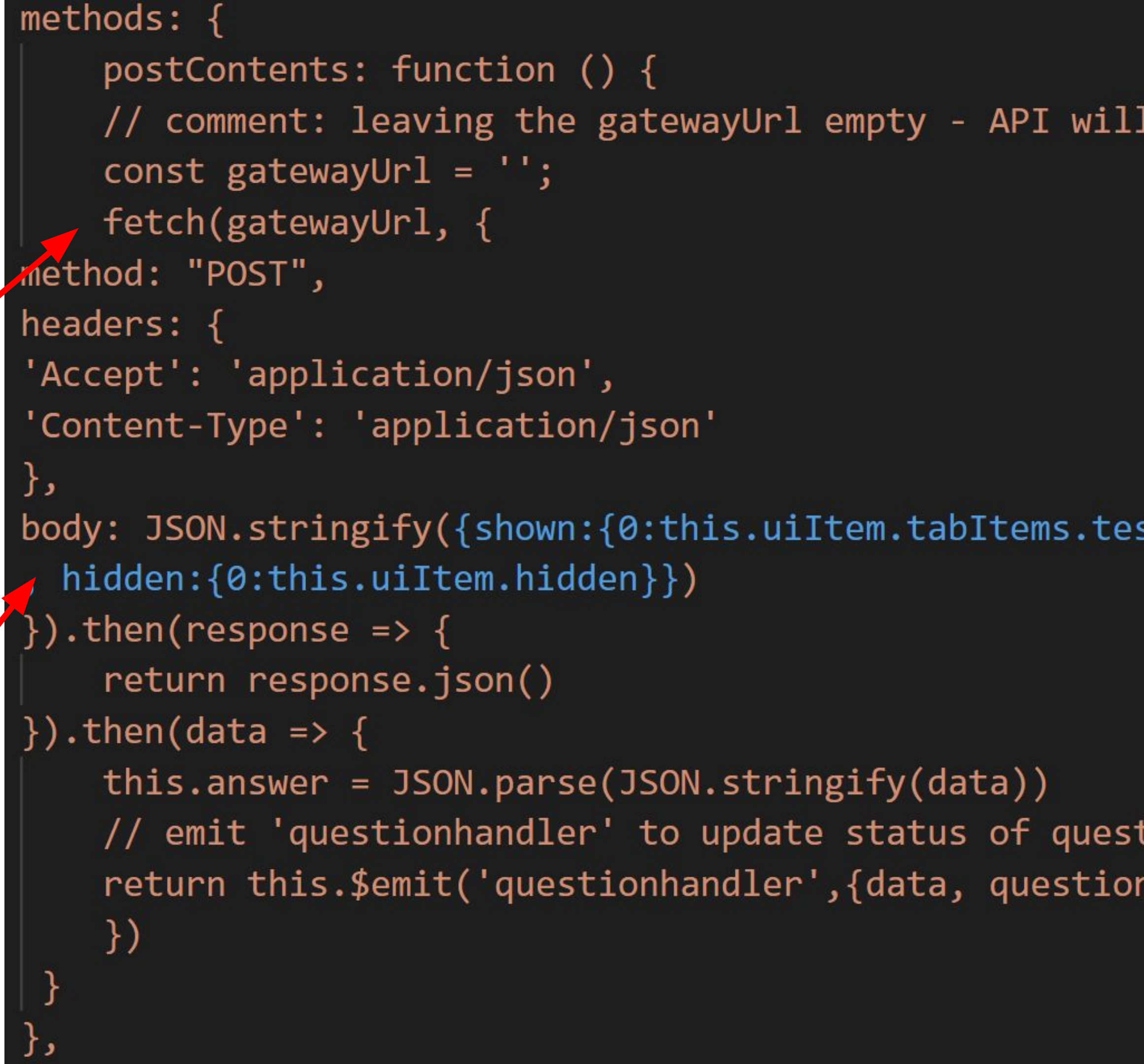


```
data: function () {
    return {
    answer:{jsonFeedback:'',htmlFeedback:'',textFeedback:'',isComplete:false},
    uiItem: this.uiItems,
    cmOptions: {
      mode: 'python',
      lineNumbers: true
    }
}
```

Demo

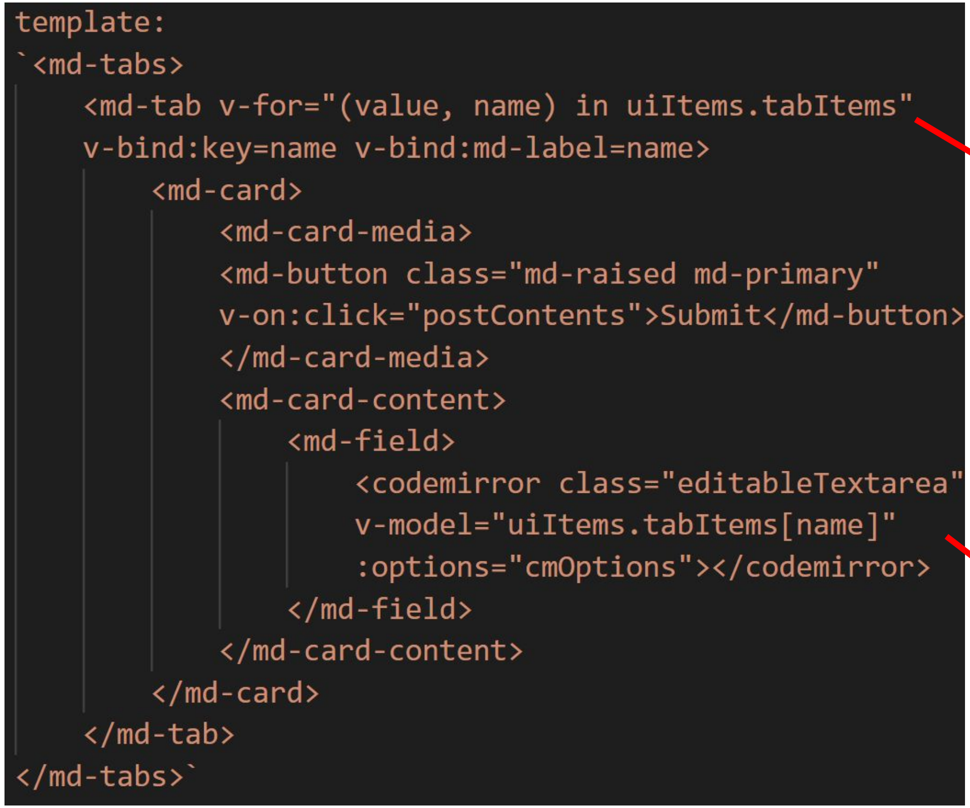
Notes about doctest-activity's postcontents method:

1) It posts back to itself when the url it is posting to is empty!
2) It posts a JSON string back to our lambda function along with some data stored in its post request body



```
methods: {
    postContents: function () {
    // comment: leaving the gatewayUrl empty - API will
    const gatewayUrl = '';
    fetch(gatewayUrl, {
method: "POST",
headers: {
'Accept': 'application/json',
'Content-Type': 'application/json'
},
body: JSON.stringify({shown:{0:this.uiItem.tabItems.tes
  hidden:{0:this.uiItem.hidden}})
}).then(response => {
    return response.json()
}).then(data => {
    this.answer = JSON.parse(JSON.stringify(data))
    // emit 'questionhandler' to update status of quest
    return this.$emit('questionhandler',{data, questior
    })
  }
},
```

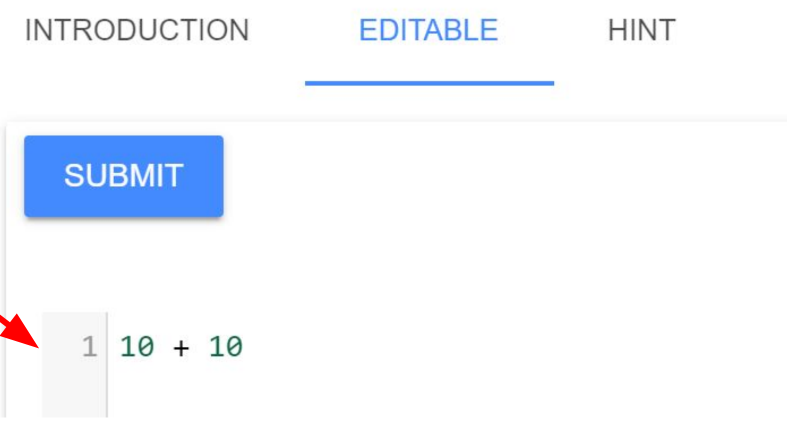
Demo



```
template:
`<md-tabs>
    <md-tab v-for="(value, name) in uiItems.tabItems"
    v-bind:key=name v-bind:md-label=name>
        <md-card>
            <md-card-media>
            <md-button class="md-raised md-primary"
            v-on:click="postContents">Submit</md-button>
            </md-card-media>
            <md-card-content>
                <md-field>
                    <codemirror class="editableTextarea"
                    v-model="uiItems.tabItems[name]"
                    :options="cmOptions"></codemirror>
                </md-field>
            </md-card-content>
        </md-card>
    </md-tab>
</md-tabs>`
```

Notes about doctest-activity's template

1) It loops through the key-value pairs in uiItem.tabItems to make tabs for each of the key-value pairs
2) Each tab displays a codemirror editable textarea



Two Files

Keep all of the lambda code, including the GUI, in a single file (lambda_handler.py). (example)

Pros:

- Collaborators can simply create a new lambda function and copy the latest code on Github into the lambda editor and save their new function.
- You can develop the GUI (index.html) separately as a standalone file.

Cons:

- All of the GUI-handling code will be in a single index.html file.

Example: https://github.com/Ourstress/minimalCodeLambdaFunction/tree/split-1file-internally

Multiple Files (without API proxy)

Split up your GUI into multiple files, such as

- styles.css / main.js etc
- Example below splits vue script into vueApp.js & doctestComponent.js

Pros:

- The GUI can be split into separate parts for easier maintainability
- No need to setup API gateway for proxy integration

Cons:

- To incorporate the split up files into the main html page, the example below used a clumsy string replacement method

Example: https://github.com/Ourstress/minimalCodeLambdaFunction/tree/split-files-internally

Proxied API

Configure the API as a proxy so that you can return different files (index.html, main.js, app.css, etc.) on GETs to your function. (tutorial) (example)

Pros:

- Your GUI can consist of many different html, js, and css files.

Cons:

- Your collaborators will have to configure their API as a proxy to pass through the complete requested route (/, /index.html, /main.js, /app.css) requested by the web browser as it tries to load the GUI files.

Github Pages

Host the GUI from Github using Github pages. (tutorial) (example)

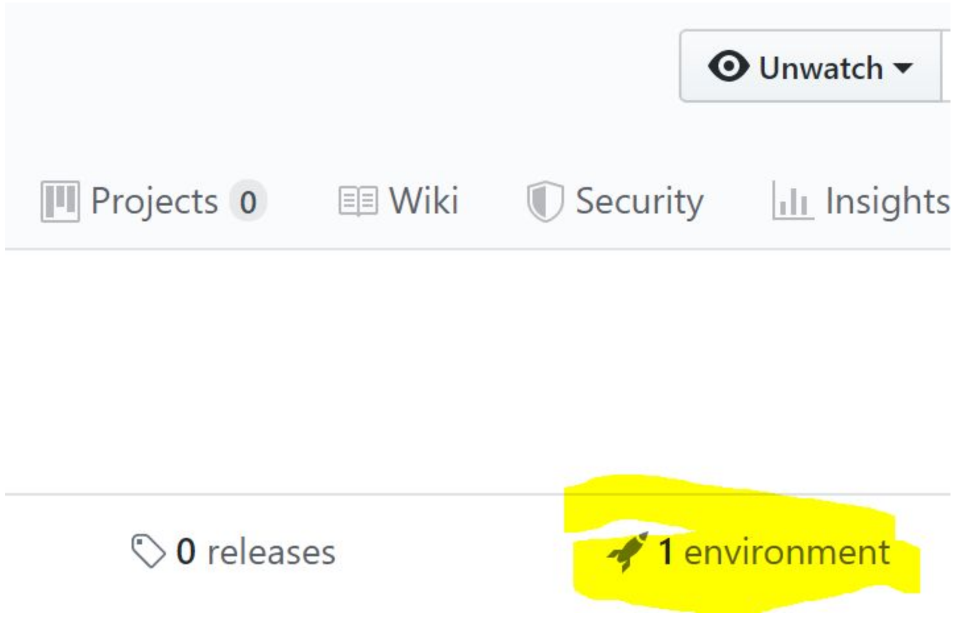
Pros:

- Since your GUI file are already hosted on Github, you can just configure Github to serve your master branch as a web page.
- All collaborators could fork the repo, edit the GUI files, and see their changes on their Github Pages-hosted fork before submitting a pull request to merge in their own changes. This would enable collaborators to work on the GUI without using AWS Lambda.

Cons:

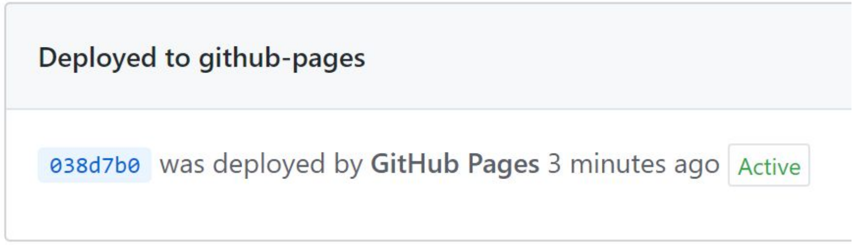
- The GUI will have to be hard coded to POST back to a specific deployed version of the Lambda code.
- The Lambda function will have to support CORS to allow the web pages downloaded from other domains (Github Pages) to access data fetched from the lambda function.

Github Pages



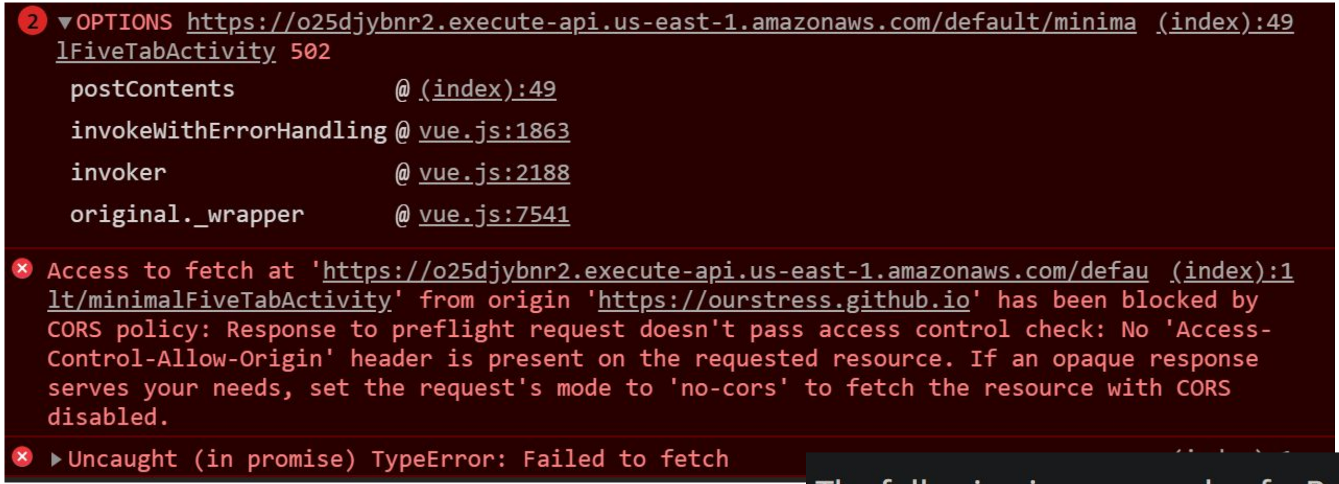
View deployment status of github pages under the environment tab of your github repo

<<<


Deployed to github-pages

038d7b0 was deployed by **GitHub Pages** 3 minutes ago Active

Github Pages



```
2 ▼OPTIONS https://o25djybnr2.execute-api.us-east-1.amazonaws.com/default/minima  (index):49
lFiveTabActivity 502
  postContents           @ (index):49
  invokeWithErrorHandling @ vue.js:1863
  invoker                @ vue.js:2188
  original._wrapper      @ vue.js:7541
```

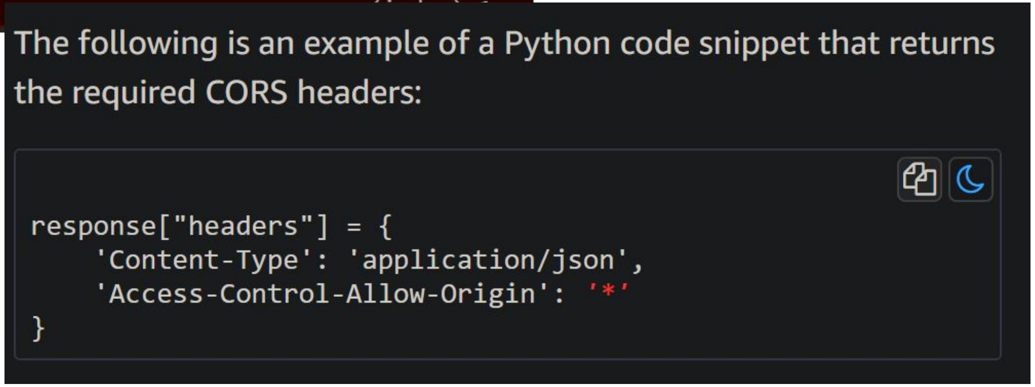
```
Access to fetch at 'https://o25djybnr2.execute-api.us-east-1.amazonaws.com/defau  (index):1
lt/minimalFiveTabActivity' from origin 'https://ourstress.github.io' has been blocked by
CORS policy: Response to preflight request doesn't pass access control check: No 'Access-
Control-Allow-Origin' header is present on the requested resource. If an opaque response
serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS
disabled.
```

```
▶Uncaught (in promise) TypeError: Failed to fetch
```

Error message due to CORS on posting to AWS Lambda function

Fix the issue by returning CORS headers when posting response back to the browser

Example



The following is an example of a Python code snippet that returns the required CORS headers:

```python
response["headers"] = {
    'Content-Type': 'application/json',
    'Access-Control-Allow-Origin': '*'
}
```

SAM Template

Add a SAM template to automate the configuration, packaging, and deployment of all files. (tutorial) (example)

Pros:

- Collaborators can clone the latest code into AWS Cloud9 and then execute “sam deploy” to create their own copy of the API and lambda function.

Cons:

- Users will need to log in to Cloud9 to use sam or will need to configure their local systems to use sam.

SAM Template - through Cloud9

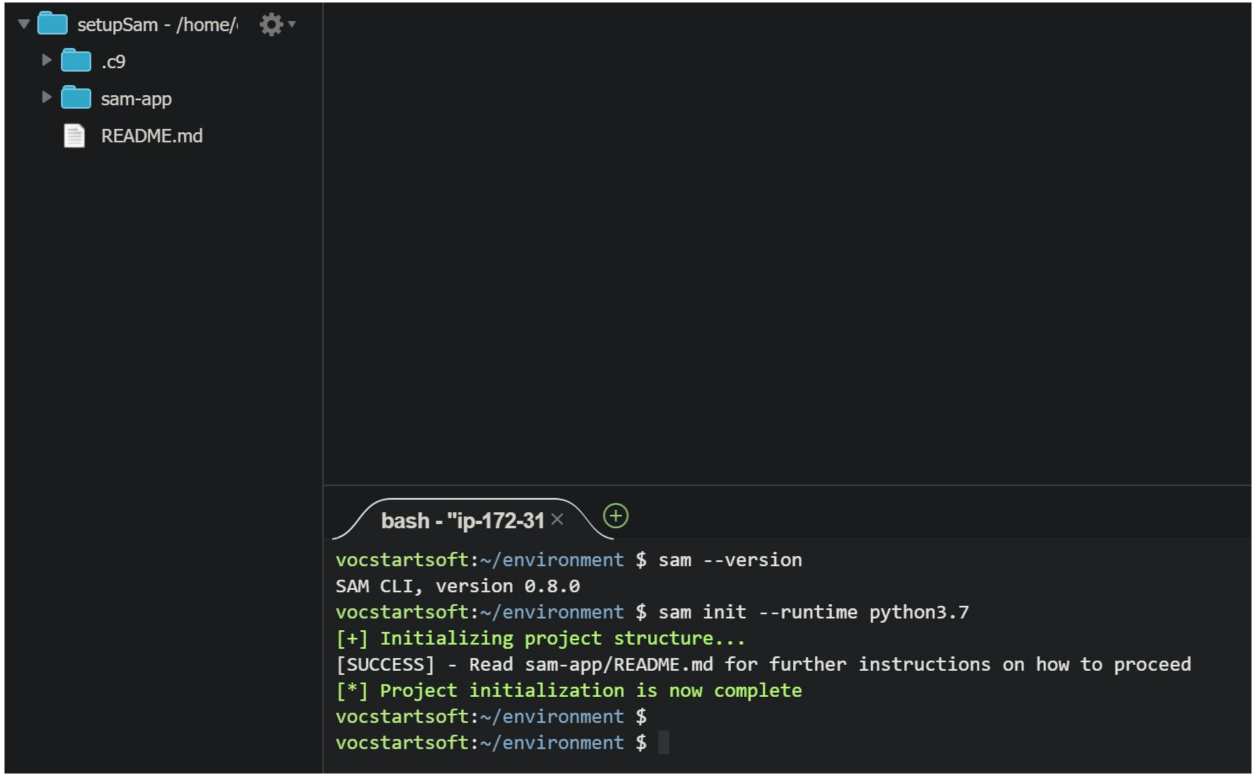
One way to get started using SAM is through cloud9

Few weeks earlier, cloud9 didn't have the SAM version that could initialise python 3.7 runtime but now it has updated to v0.8.0 that can do so!

Run this command

```
sam init --runtime python3.7
```

to creates a directory named sam-app that contains the some files we can build on top of



```
vocstartsoft:~/environment $ sam --version
SAM CLI, version 0.8.0
vocstartsoft:~/environment $ sam init --runtime python3.7
[+] Initializing project structure...
[SUCCESS] - Read sam-app/README.md for further instructions on how to proceed
[*] Project initialization is now complete
vocstartsoft:~/environment $
vocstartsoft:~/environment $
```

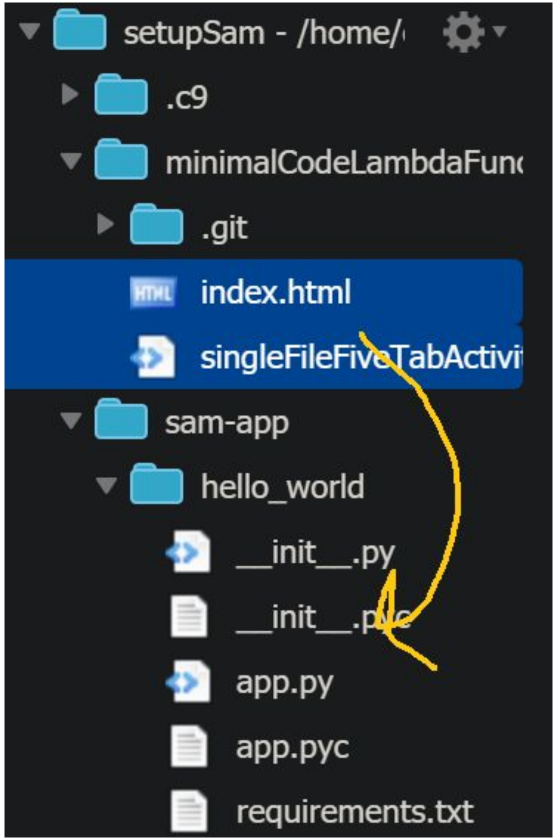
SAM Template - through Cloud9

2) Customise your lambda function

Next, you could modify the default app.py with some lambda function code you've written.

Alternatively, git clone your github repo. If you are git cloning a branch, consider using git clone --branch BRANCH-NAME.

After git cloning, move your files over to sam-app's hello world function and change your lambda function's filename to app.py



SAM Template - through Cloud9

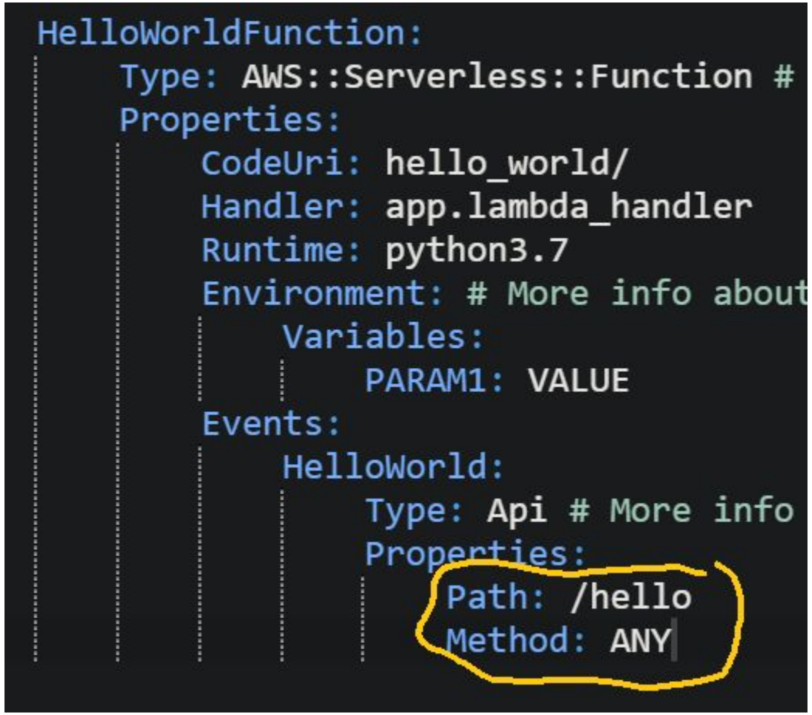
2) Customise your lambda function

Modify the template.yaml file to set Method to **ANY**

- Otherwise, the default only allows GET and not POST. You can retrieve your index.html but get 403 unauthorised response when posting back

You can also change the Path

- The default is /hello but you can change it to /



```
HelloWorldFunction:
    Type: AWS::Serverless::Function #
    Properties:
        CodeUri: hello_world/
        Handler: app.lambda_handler
        Runtime: python3.7
        Environment: # More info about
            Variables:
                PARAM1: VALUE
        Events:
            HelloWorld:
                Type: Api # More info
                Properties:
                    Path: /hello
                    Method: ANY
```

SAM Template - through Cloud9

3) **sam build**

OR **sam build --use-container** if sam build fails because Cloud9 uses Python2.7 runtime

sam build iterates through the functions in your application, looks for a manifest file (such as requirements.txt) that contains the dependencies, and automatically creates deployment artifacts

This means to install Pandas library for instance, we just need to specify “pandas” in requirements.txt!

Another point to note: **use python 3.6 runtime** if you plan to deploy your lambda function using AWS-SAM-CLI

SAM Template - through Cloud9

Fill in the parts in bold below, and you are done!

4) sam package --output-template packaged.yaml --s3-bucket **YOUR-BUCKET-NAME**

5) aws cloudformation deploy --template-file /home/ec2-user/environment/sam-app/packaged.yaml --capabilities CAPABILITY_IAM --stack-name **YOUR-STACK-NAME**

After deploying on cloud9, you can view the results!

SAM Template - explanation

Globals read the docs:

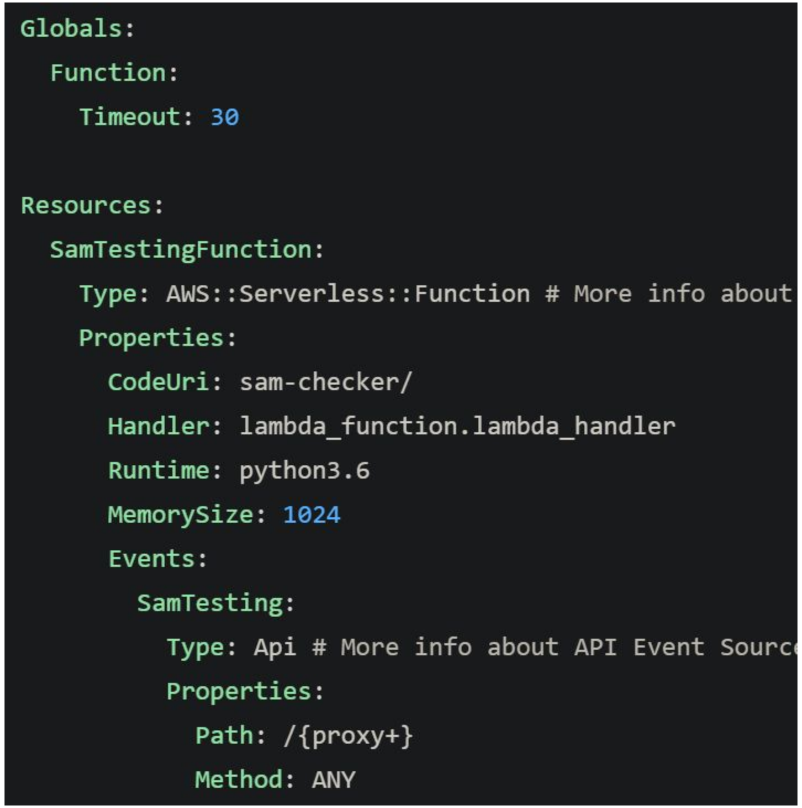
- Resources in a SAM template tend to have shared configuration such as Runtime, Memory, VPC Settings, Environment Variables, Cors, etc
- Instead of duplicating this information in every resource, you can write them once in the Globals section and let all resources inherit it.

Resources Read the docs on resources:

- In the resources section, we can create API gateways, lambda layers, lambda functions and dynamoDB tables

Events Read the docs on events:

- AWS services that can act as event sources for our lambda functions can be specified in the events section



```
Globals:
  Function:
    Timeout: 30

Resources:
  SamTestingFunction:
    Type: AWS::Serverless::Function # More info about
    Properties:
      CodeUri: sam-checker/
      Handler: lambda_function.lambda_handler
      Runtime: python3.6
      MemorySize: 1024
      Events:
        SamTesting:
          Type: Api # More info about API Event Sourc
          Properties:
            Path: /{proxy+}
            Method: ANY
```

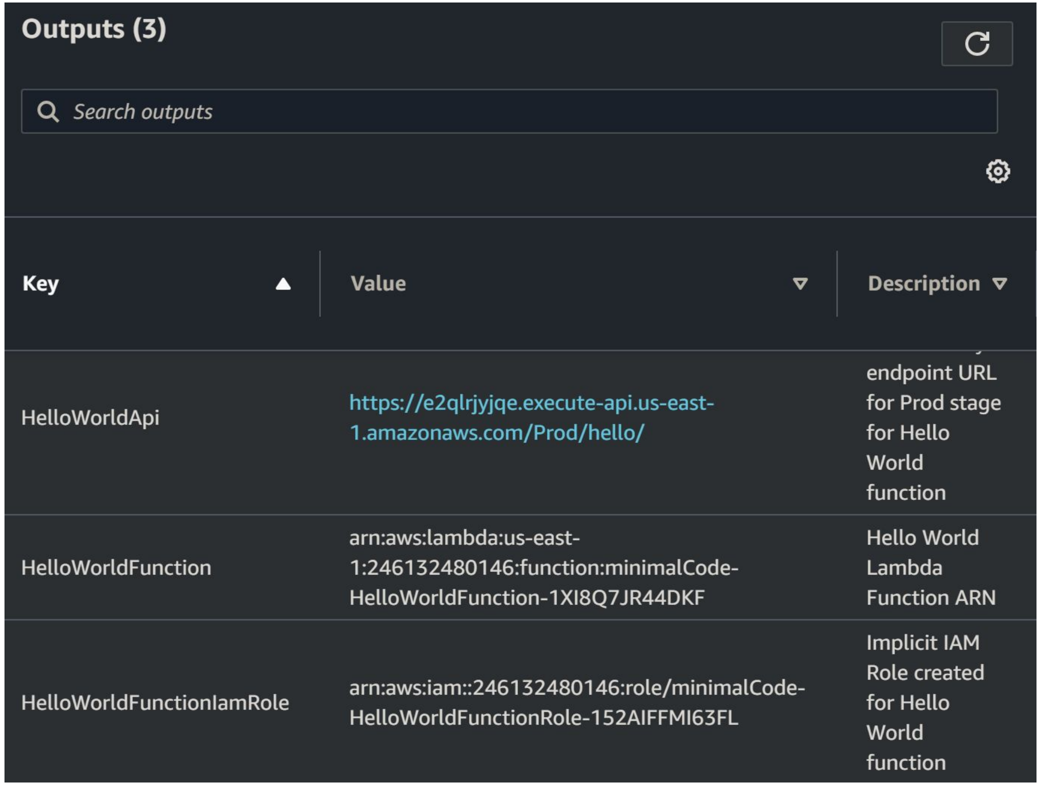
SAM Template - explanation

The optional Outputs section declares output values

- that we can import into other stacks (to create cross-stack references)
- return in response (to describe stack calls)
- view on the AWS CloudFormation console

For more info about outputs, click here

And this was the output from the SAM exercise earlier -->



Key	Value	Description
HelloWorldApi	https://e2qlrjyjqe.execute-api.us-east-1.amazonaws.com/Prod/hello/	endpoint URL for Prod stage for Hello World function
HelloWorldFunction	arn:aws:lambda:us-east-1:246132480146:function:minimalCode-HelloWorldFunction-1XI8Q7JR44DKF	Hello World Lambda Function ARN
HelloWorldFunctionIamRole	arn:aws:iam::246132480146:role/minimalCode-HelloWorldFunctionRole-152AIFFMI63FL	Implicit IAM Role created for Hello World function

Working on AWS notes

- We can always reuse the same s3 bucket and stack
- When we modify our lambda function and then sam build / package / deploy to the same stack, it updates the existing stack and our lambda function url etc stays the same so users aren't inconvenienced
- Always use the region **us-east-1**
- For multiple events like two api gateways, see https://github.com/awslabs/serverless-application-model/issues/209

Github Action

Configure a Github action to redeploy the latest code on commits to a repository. (tutorial) (example)

Pros:

- Automatically deploys the latest code to AWS everytime a commit is made to the Github repository.

Cons:

- You have to configure Github actions and AWS SAM or AWS Cloudformation

Guide to github actions

Github Actions with deployment using SAM

Github actions comprises workflows and actions.

Setup a github workflow:

1) Add a folder called .github to the root directory of your github repo with an inner folder called workflows containing an **empty main.yaml file**

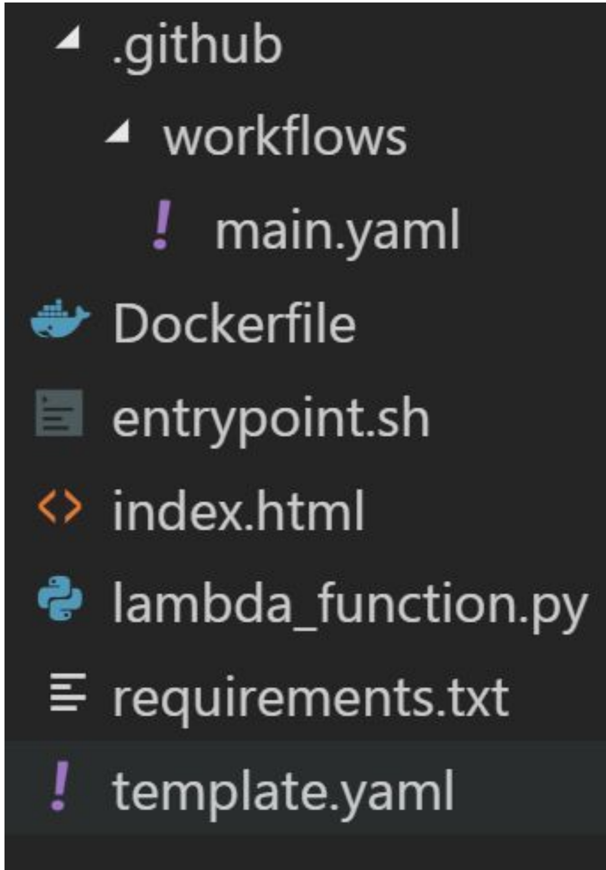
Setup a github action to deploy lambda function to AWS using SAM

2) Add an **empty dockerfile** and an **empty entrypoint.sh** to the root directory of your github repo

Setup files needed by AWS-SAM-CLI

3) Add an **empty template.yaml** and an **empty requirements.txt** to the root directory of your github repo (see purpose of requirements.txt)

Note: these files except .github above don't necessarily need to be in the root directory, you can put them in other folders once you are comfortable with the steps. Also, github Actions refers to the whole shebang (note the capital A is capitalised) whereas an individual task is called action in lowercase.

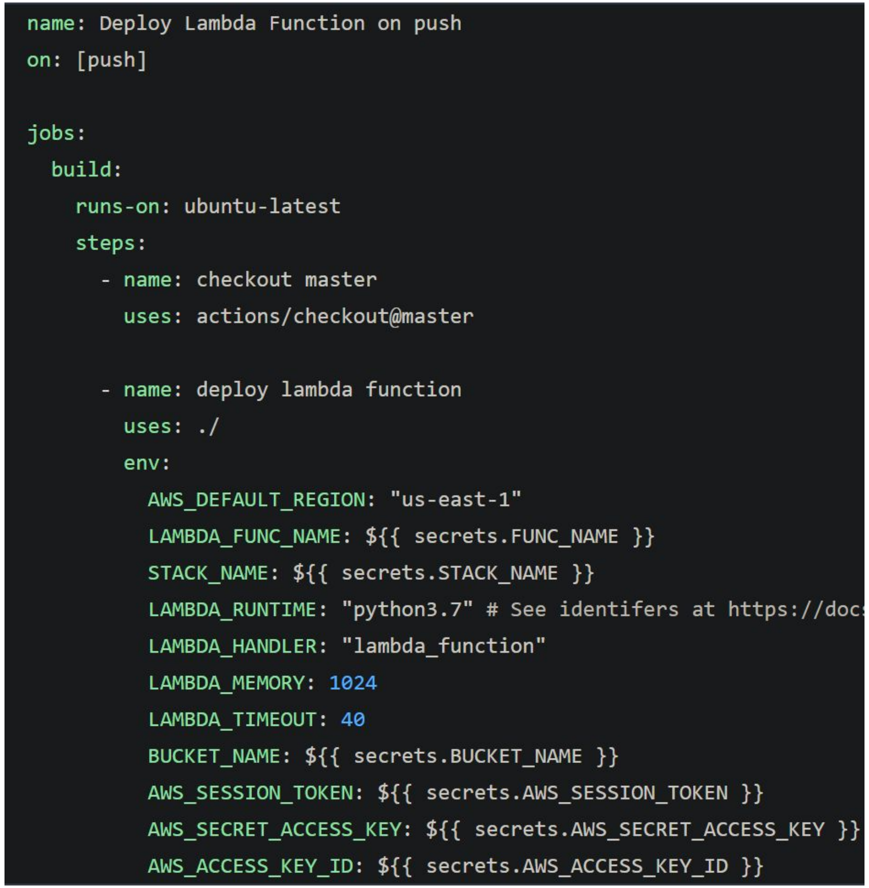


```
.github
  workflows
    main.yaml
Dockerfile
entrypoint.sh
index.html
lambda_function.py
requirements.txt
template.yaml
```

Github Actions with deployment using SAM

Github workflows main.yaml file

- Line by line walkthrough of workflow code
- Example code

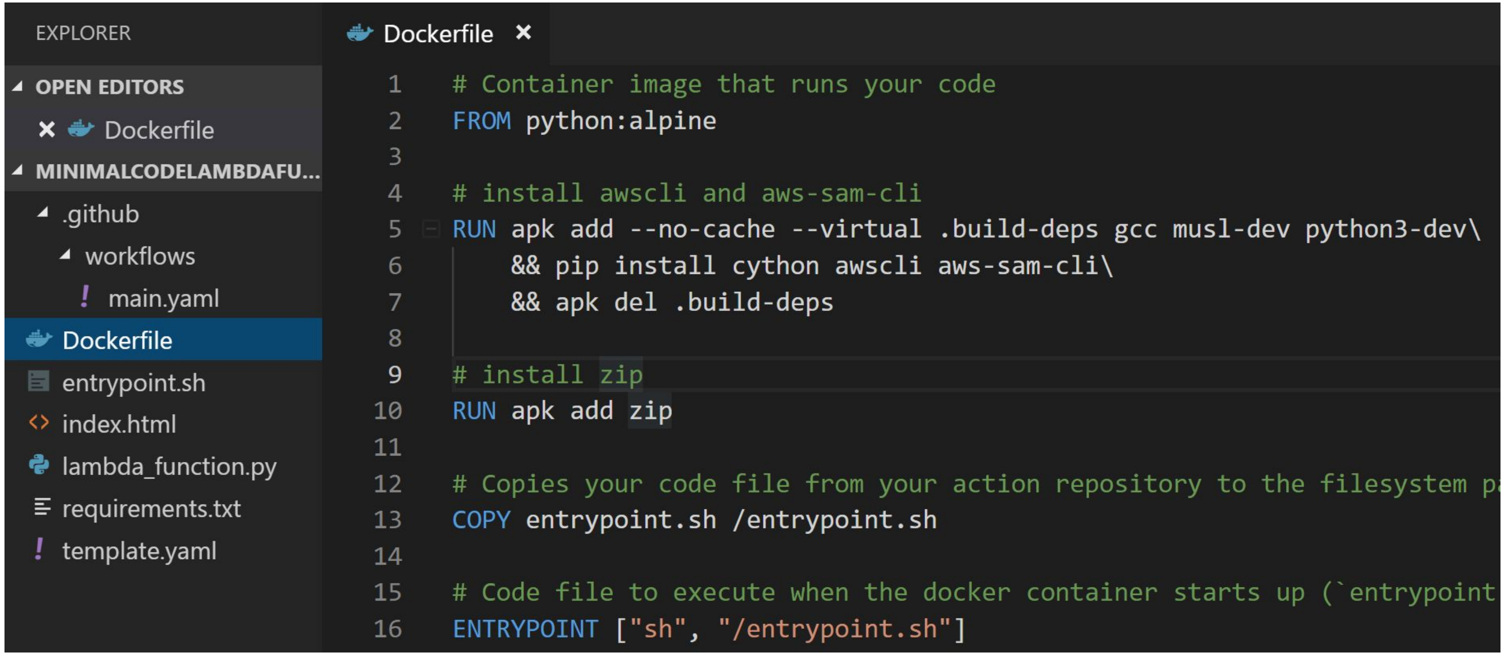


```yaml
name: Deploy Lambda Function on push
on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: checkout master
        uses: actions/checkout@master

      - name: deploy lambda function
        uses: ./
        env:
          AWS_DEFAULT_REGION: "us-east-1"
          LAMBDA_FUNC_NAME: ${{ secrets.FUNC_NAME }}
          STACK_NAME: ${{ secrets.STACK_NAME }}
          LAMBDA_RUNTIME: "python3.7" # See identifers at https://doc
          LAMBDA_HANDLER: "lambda_function"
          LAMBDA_MEMORY: 1024
          LAMBDA_TIMEOUT: 40
          BUCKET_NAME: ${{ secrets.BUCKET_NAME }}
          AWS_SESSION_TOKEN: ${{ secrets.AWS_SESSION_TOKEN }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
```

Github Actions with deployment using SAM

Dockerfile

- Base image from alpine already has python installed
- We need gcc, musl-dev, python3-dev and cython to install awscli & aws-sam-cli
- [Line by line walkthrough of Dockerfile code]()



```dockerfile
# Container image that runs your code
FROM python:alpine

# install awscli and aws-sam-cli
RUN apk add --no-cache --virtual .build-deps gcc musl-dev python3-dev\
    && pip install cython awscli aws-sam-cli\
    && apk del .build-deps

# install zip
RUN apk add zip

# Copies your code file from your action repository to the filesystem p
COPY entrypoint.sh /entrypoint.sh

# Code file to execute when the docker container starts up (`entrypoint
ENTRYPOINT ["sh", "/entrypoint.sh"]
```

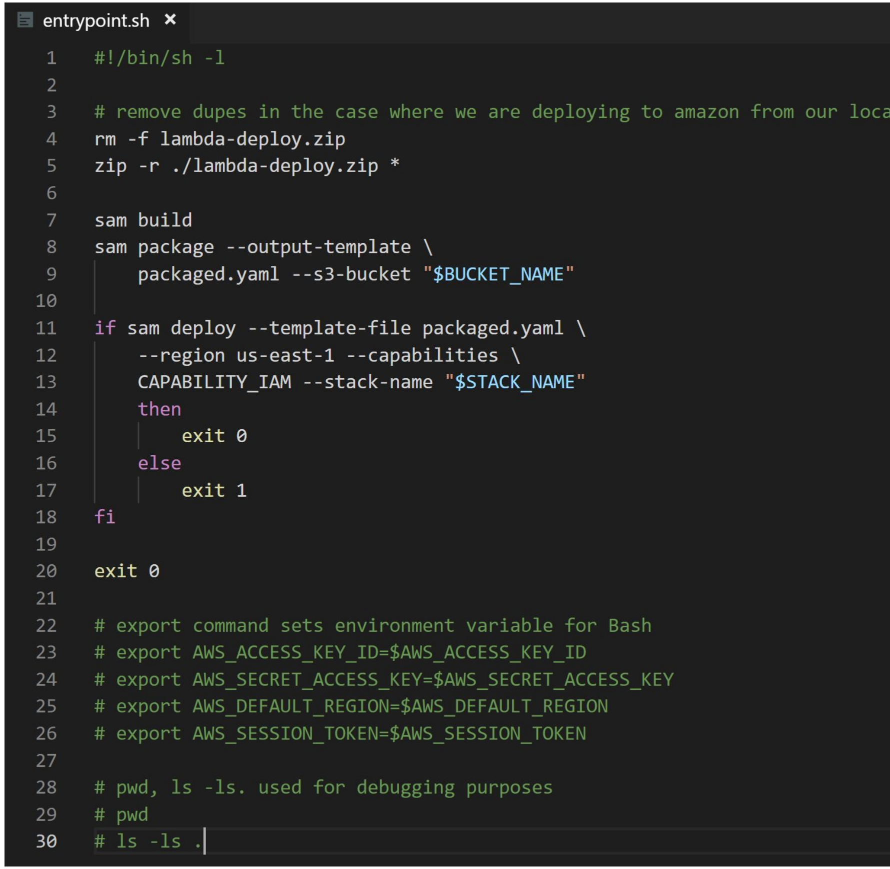
Github Actions with deployment using SAM

entrypoint.sh

- rm -f lambda-deploy.zip (deployment zip file) in case user deploys locally and ends up creating multiple zip files
- Then we zip up the current working directory as lambda-deploy.zip
- sam build, sam package and sam deploy let us deploy our lambda function on AWS
- The if else block is used to report error if the final step fails
- Example code

Notes - commented out the following:

- Exporting environment variables that AWS-SAM-CLI requires. We can comment out those lines because the we have specified the same variables in workflow main.yaml
- Pwd and ls -ls . will log the current working directory and its contents to github actions

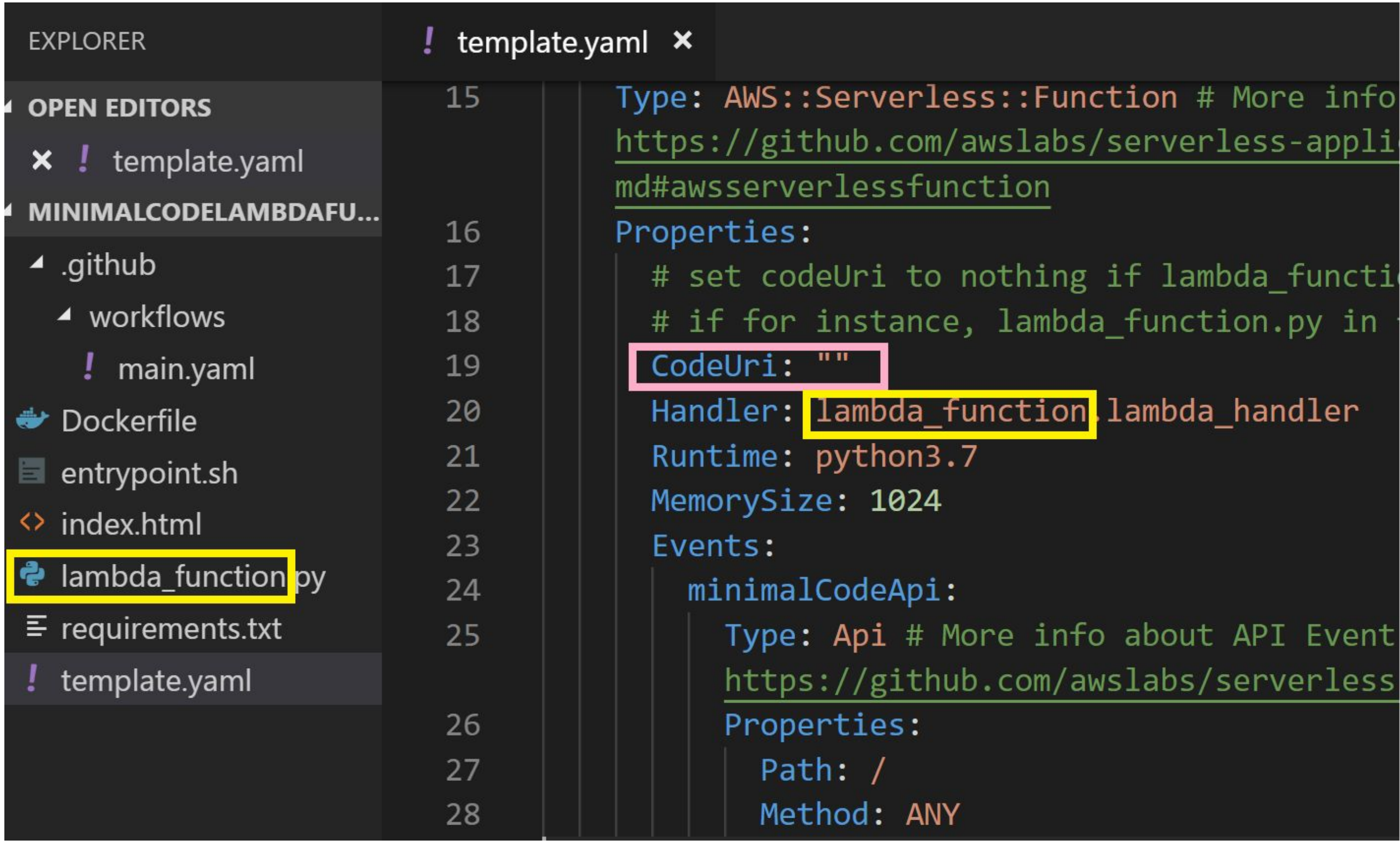

entrypoint.sh

```
#!/bin/sh -l

# remove dupes in the case where we are deploying to amazon from our loca
rm -f lambda-deploy.zip
zip -r ./lambda-deploy.zip *

sam build
sam package --output-template \
    packaged.yaml --s3-bucket "$BUCKET_NAME"

if sam deploy --template-file packaged.yaml \
    --region us-east-1 --capabilities \
    CAPABILITY_IAM --stack-name "$STACK_NAME"
    then
        exit 0
    else
        exit 1
fi

exit 0

# export command sets environment variable for Bash
# export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY_ID
# export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_ACCESS_KEY
# export AWS_DEFAULT_REGION=$AWS_DEFAULT_REGION
# export AWS_SESSION_TOKEN=$AWS_SESSION_TOKEN

# pwd, ls -ls. used for debugging purposes
# pwd
# ls -ls .
```

Github Actions with deployment using SAM

template.yaml file

- Notice codeUri specifying which folder contains the script which has the lambda handler
- The name of the file containing lambda handler must match (see yellow boxes)
- Example code
- **use python 3.6 runtime** if you plan to deploy your lambda function using AWS-SAM-CLI

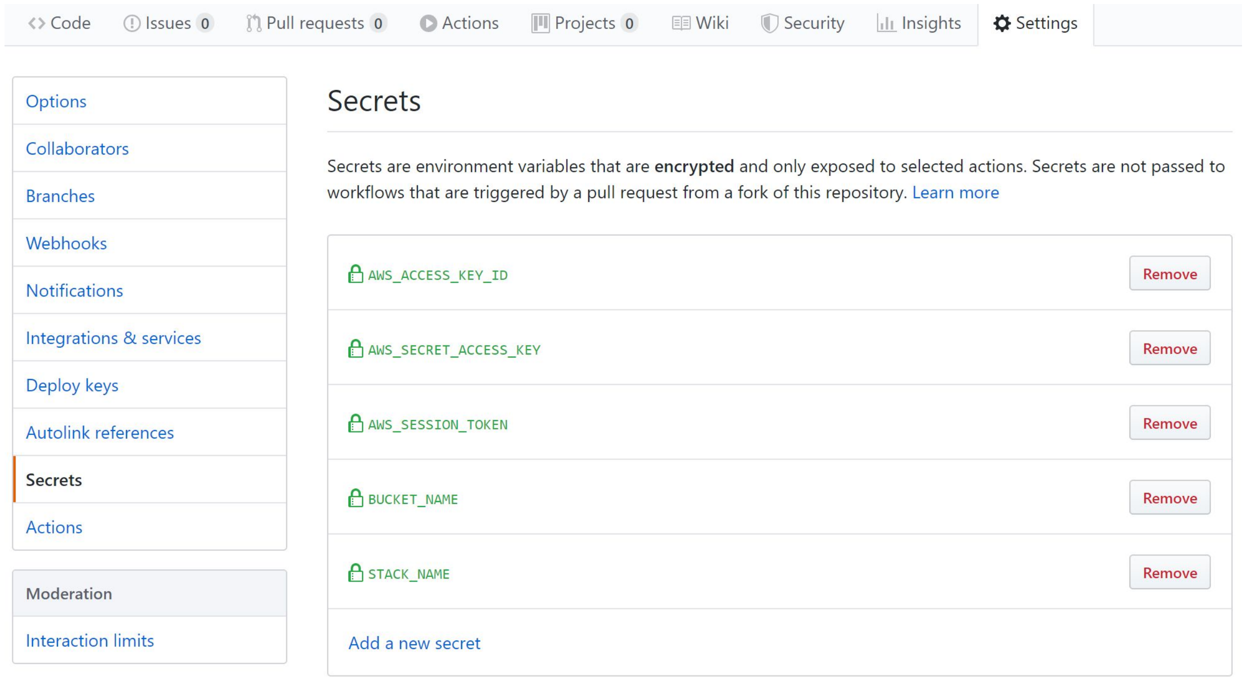


```yaml
      Type: AWS::Serverless::Function # More info
      https://github.com/awslabs/serverless-appli
      md#awsserverlessfunction
      Properties:
        # set codeUri to nothing if lambda_functi
        # if for instance, lambda_function.py in
        CodeUri: ""
        Handler: lambda_function.lambda_handler
        Runtime: python3.7
        MemorySize: 1024
        Events:
          minimalCodeApi:
            Type: Api # More info about API Event
            https://github.com/awslabs/serverless
            Properties:
              Path: /
              Method: ANY
```

Github Actions with deployment using SAM

Fill up secrets

- Navigate to your Github console. Click on settings and then click on "Secrets"



Secrets

Secrets are environment variables that are **encrypted** and only exposed to selected actions. Secrets are not passed to workflows that are triggered by a pull request from a fork of this repository. Learn more

AWS_ACCESS_KEY_ID Remove

AWS_SECRET_ACCESS_KEY Remove

AWS_SESSION_TOKEN Remove

BUCKET_NAME Remove

STACK_NAME Remove

Add a new secret

Add the following secrets -

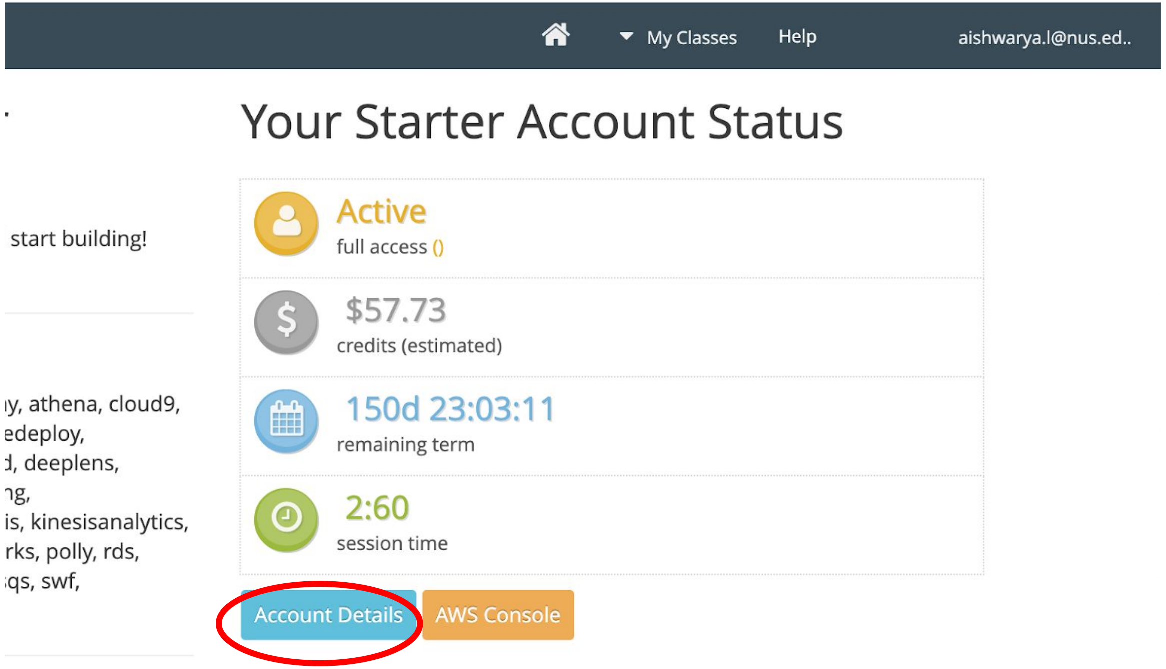
- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**
- **AWS_SESSION_TOKEN**
- **BUCKET_NAME -** The S3 bucket name with publically accessible objects
- **STACK_NAME** - name of your cloudformation stack

For AWS Educate users, credentials can be accessed as shown in the following slides

Github Actions with deployment using SAM

Fill up secrets

The values for AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_SESSION_TOKEN can be accessed in your AWS Educate console



Your Starter Account Status

start building!

Active
full access ()

$57.73
credits (estimated)

150d 23:03:11
remaining term

2:60
session time

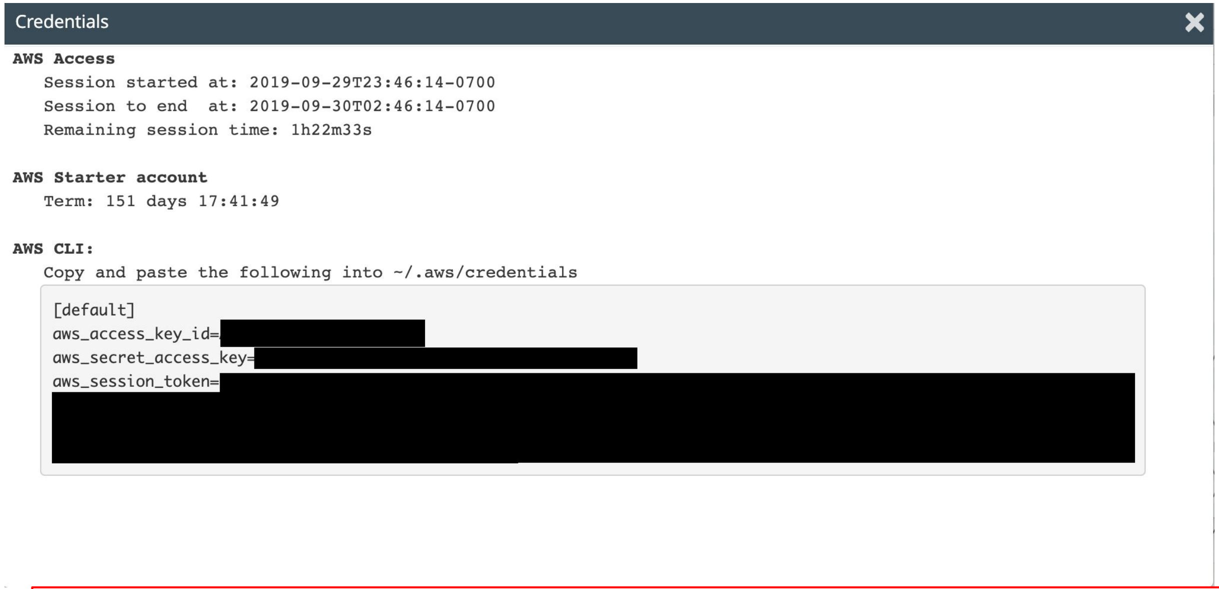
ıy, athena, cloud9,
edeploy,
ɟ, deeplens,
ng,
is, kinesisanalytics,
rks, polly, rds,
ɪqs, swf,

NOTE: For AWS Educate users, user credentials can be found here -

The credentials expire every 3 hours and must be overwritten in the GitHub console before every push for successful auto-deployment.

Github Actions with deployment using SAM

Fill up secrets



Credentials

AWS Access

```
Session started at: 2019-09-29T23:46:14-0700
Session to end  at: 2019-09-30T02:46:14-0700
Remaining session time: 1h22m33s
```

AWS Starter account

```
Term: 151 days 17:41:49
```

AWS CLI:

Copy and paste the following into ~/.aws/credentials

```
[default]
aws_access_key_id=[redacted]
aws_secret_access_key=[redacted]
aws_session_token=[redacted]
```

NOTE:

Copy the values corresponding to the keys required and add them in your Github console.

Github Actions and SAM notes

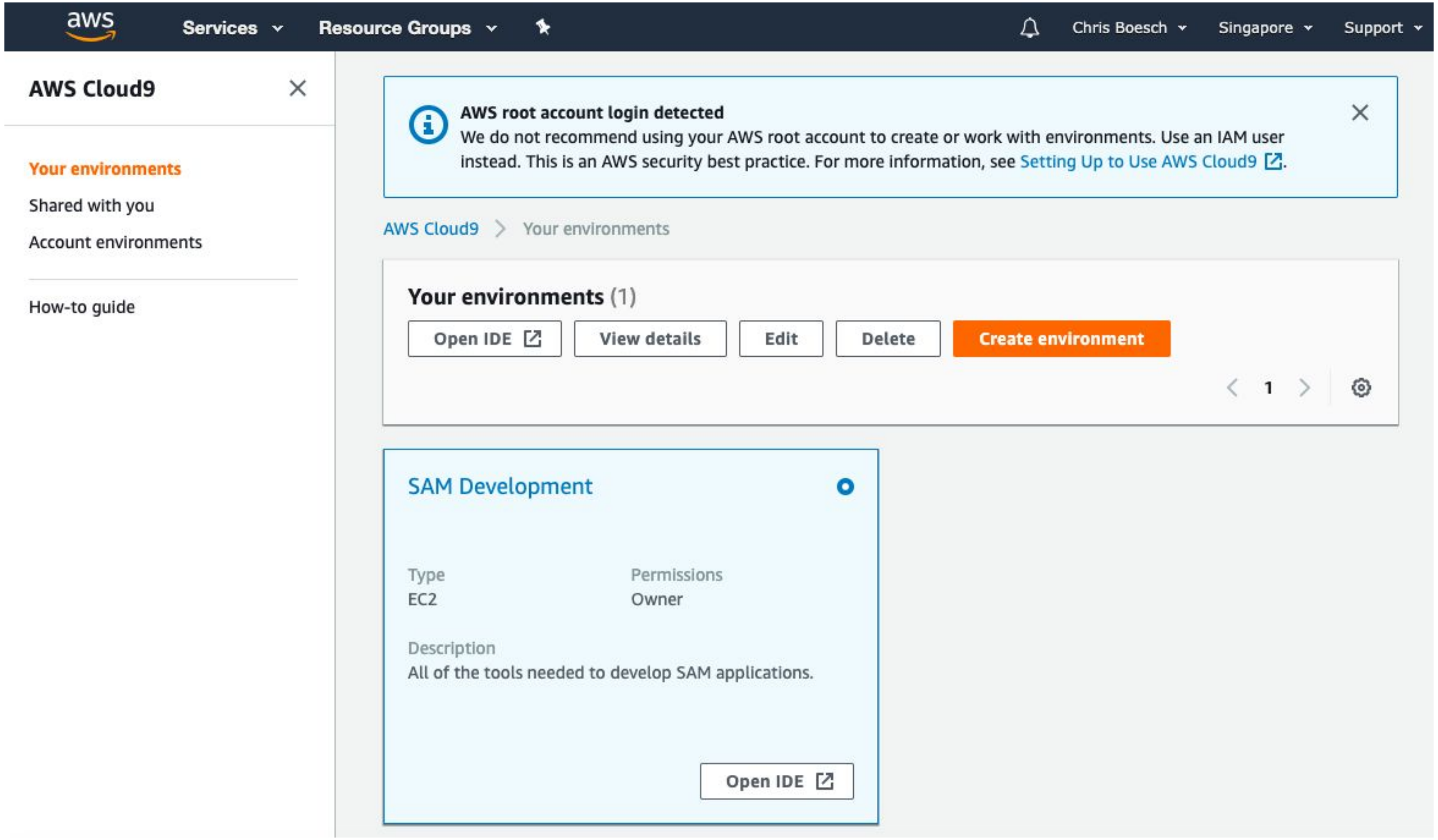
- Further resources - Github Actions slides from week7

AWS SAM Tutorial

- “The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications. ... SAM CLI provides a Lambda-like execution environment that lets you locally build, test, and debug applications defined by SAM templates. You can also use the SAM CLI to deploy your applications to AWS.”
- Work through the following tutorial to deploy an application using AWS SAM.
- Submit a link to your deployed SAM application which will log to firebase and AWS DynamoDB

Reference: https://aws.amazon.com/serverless/sam/

Create an Cloud9 Environment (link)



AWS root account login detected
We do not recommend using your AWS root account to create or work with environments. Use an IAM user instead. This is an AWS security best practice. For more information, see Setting Up to Use AWS Cloud9.

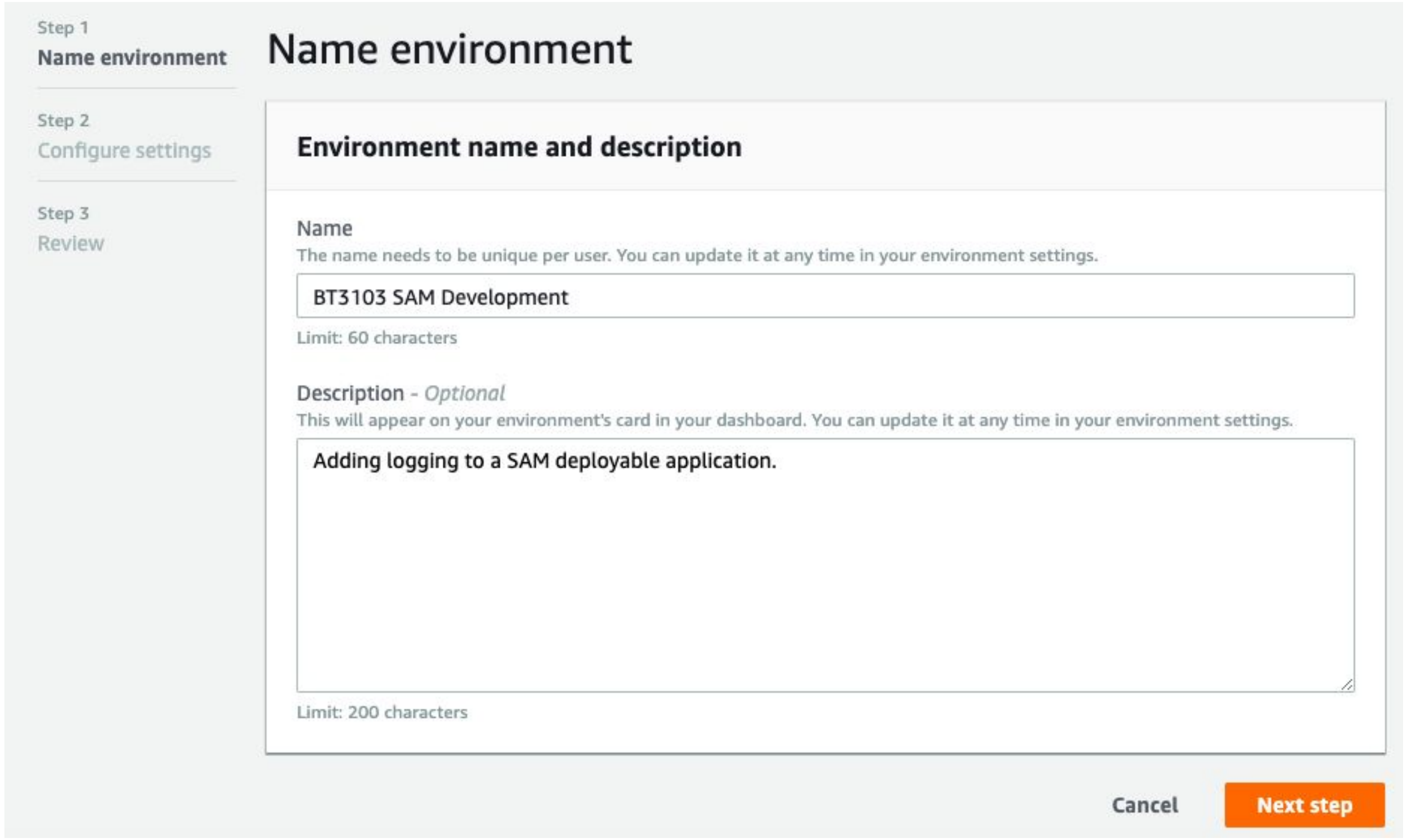
SAM Development

Type
EC2

Permissions
Owner

Description
All of the tools needed to develop SAM applications.

Add a name and description for your environment.



Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Name environment

Environment name and description

Name
The name needs to be unique per user. You can update it at any time in your environment settings.

BT3103 SAM Development

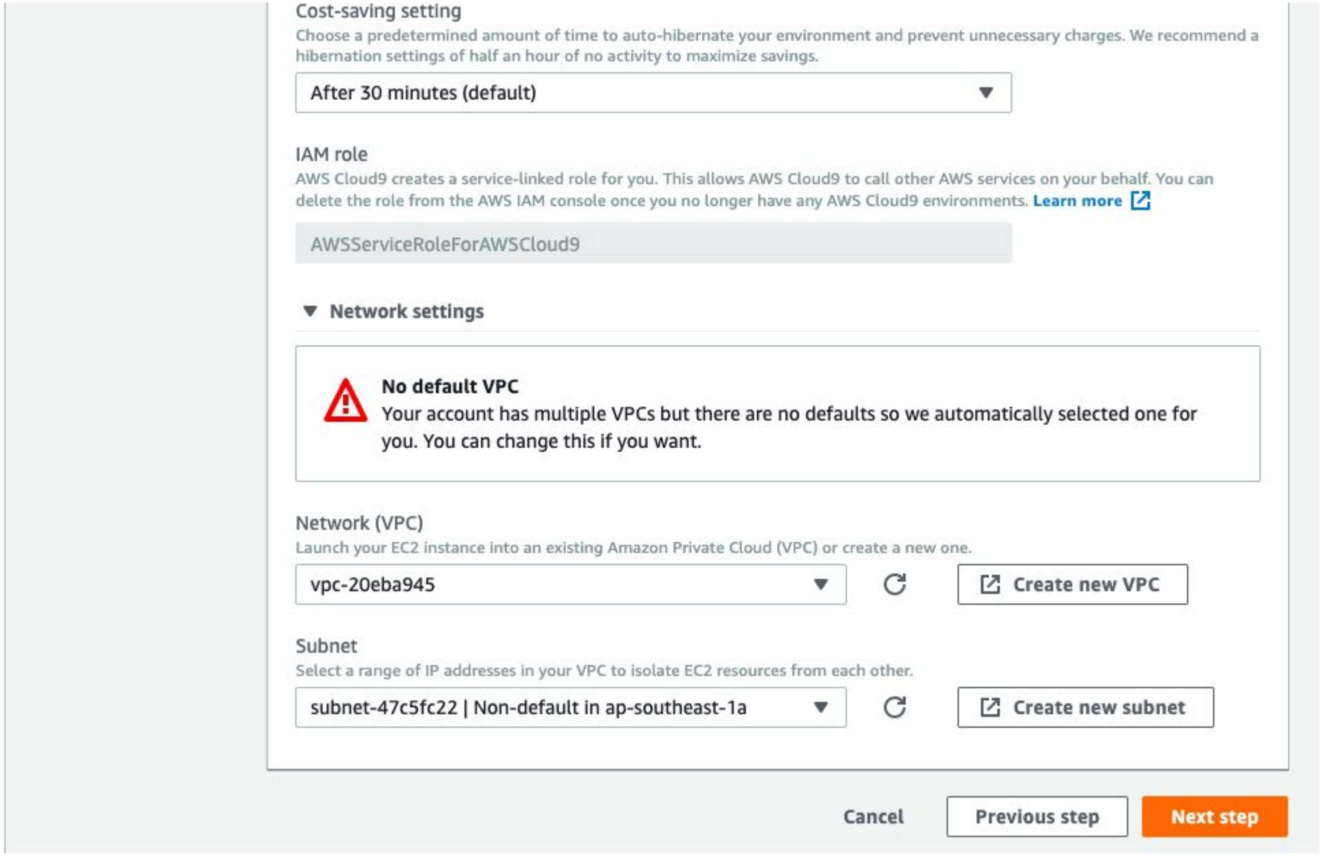
Limit: 60 characters

Description - *Optional*
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.

Adding logging to a SAM deployable application.

Limit: 200 characters

Accept all the defaults



Cost-saving setting

Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.

After 30 minutes (default)

IAM role

AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. Learn more

AWSServiceRoleForAWSCloud9

Network settings

No default VPC
Your account has multiple VPCs but there are no defaults so we automatically selected one for you. You can change this if you want.

Network (VPC)

Launch your EC2 instance into an existing Amazon Private Cloud (VPC) or create a new one.

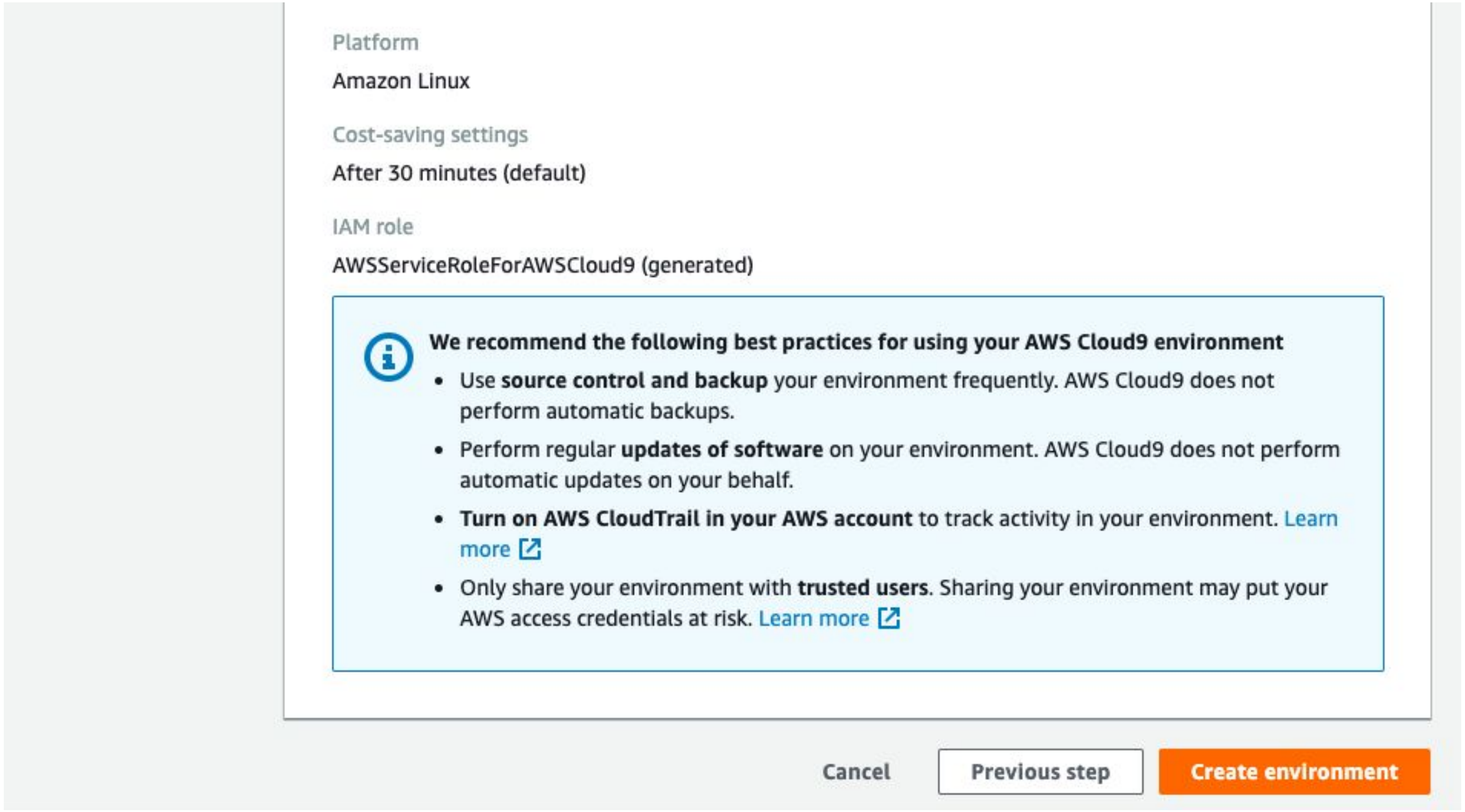
vpc-20eba945

Subnet

Select a range of IP addresses in your VPC to isolate EC2 resources from each other.

subnet-47c5fc22 | Non-default in ap-southeast-1a

Again, accept the defaults



Platform

Amazon Linux

Cost-saving settings

After 30 minutes (default)

IAM role

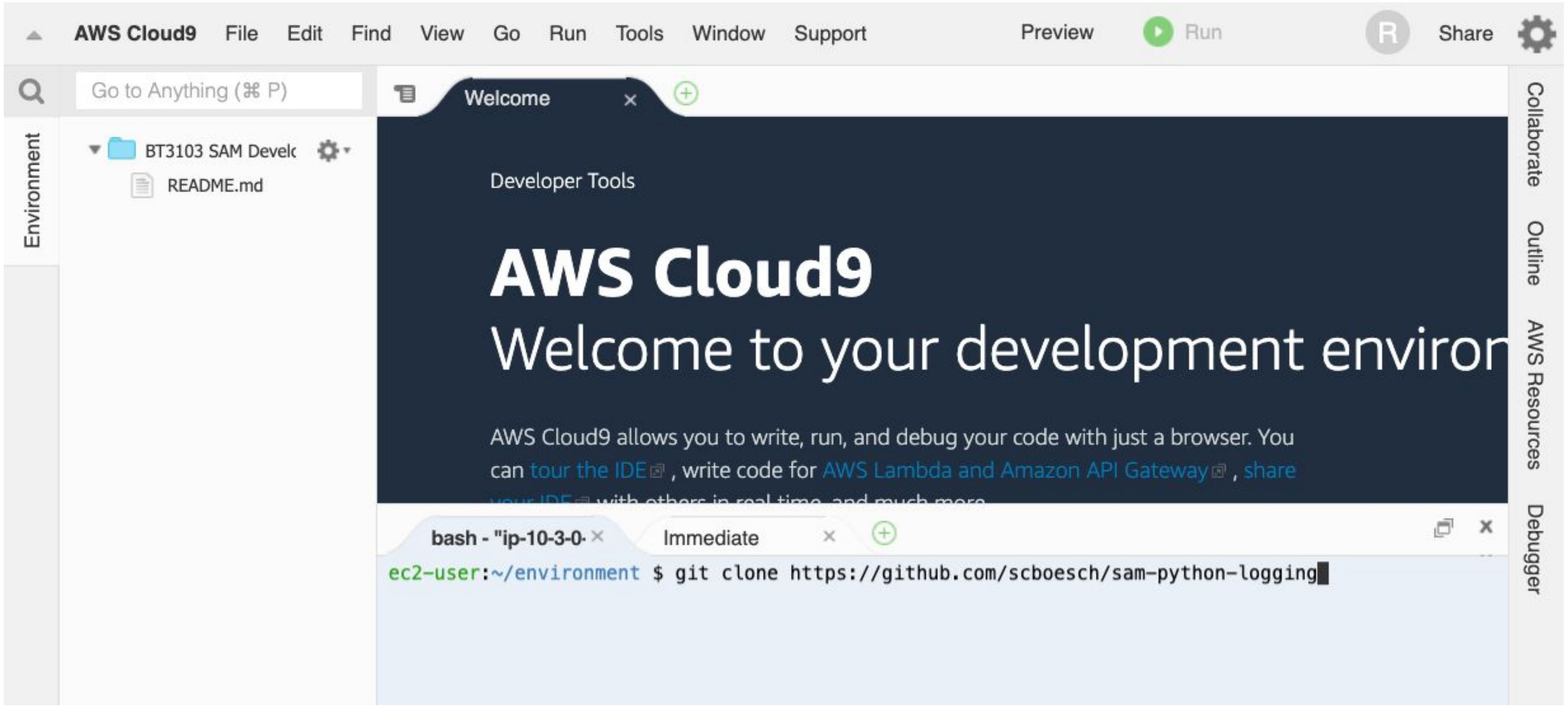
AWSServiceRoleForAWSCloud9 (generated)

We recommend the following best practices for using your AWS Cloud9 environment

- Use **source control and backup** your environment frequently. AWS Cloud9 does not perform automatic backups.
- Perform regular **updates of software** on your environment. AWS Cloud9 does not perform automatic updates on your behalf.
- **Turn on AWS CloudTrail in your AWS account** to track activity in your environment. Learn more
- Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. Learn more

Use the bash window to clone the repo into your new environment.

- git clone https://github.com/**scboesch**/sam-python-logging



Developer Tools

AWS Cloud9

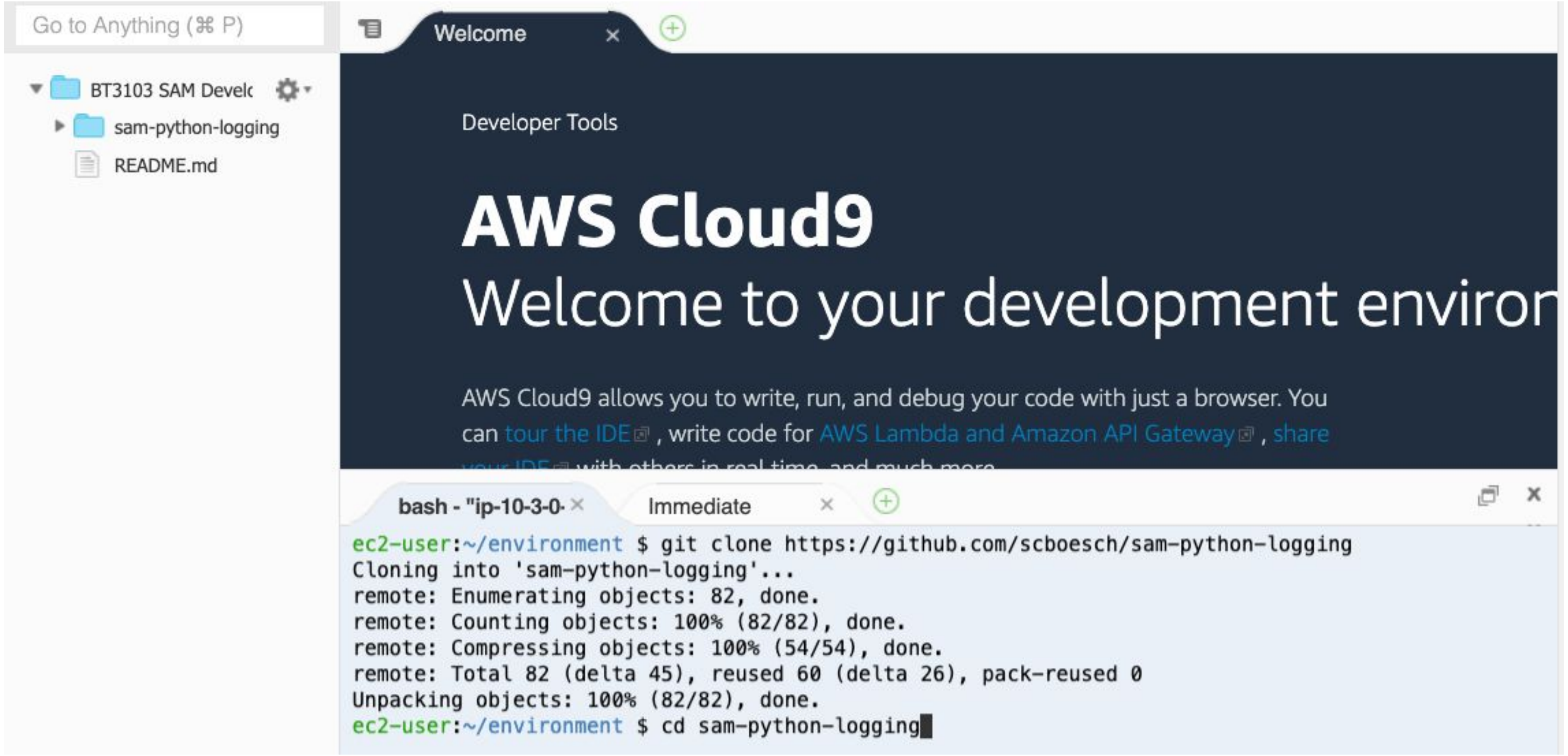
Welcome to your development enviror

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.

```
ec2-user:~/environment $ git clone https://github.com/scboesch/sam-python-logging
```

Change directory to the new cloned repo directory.

- cd sam-python-logging

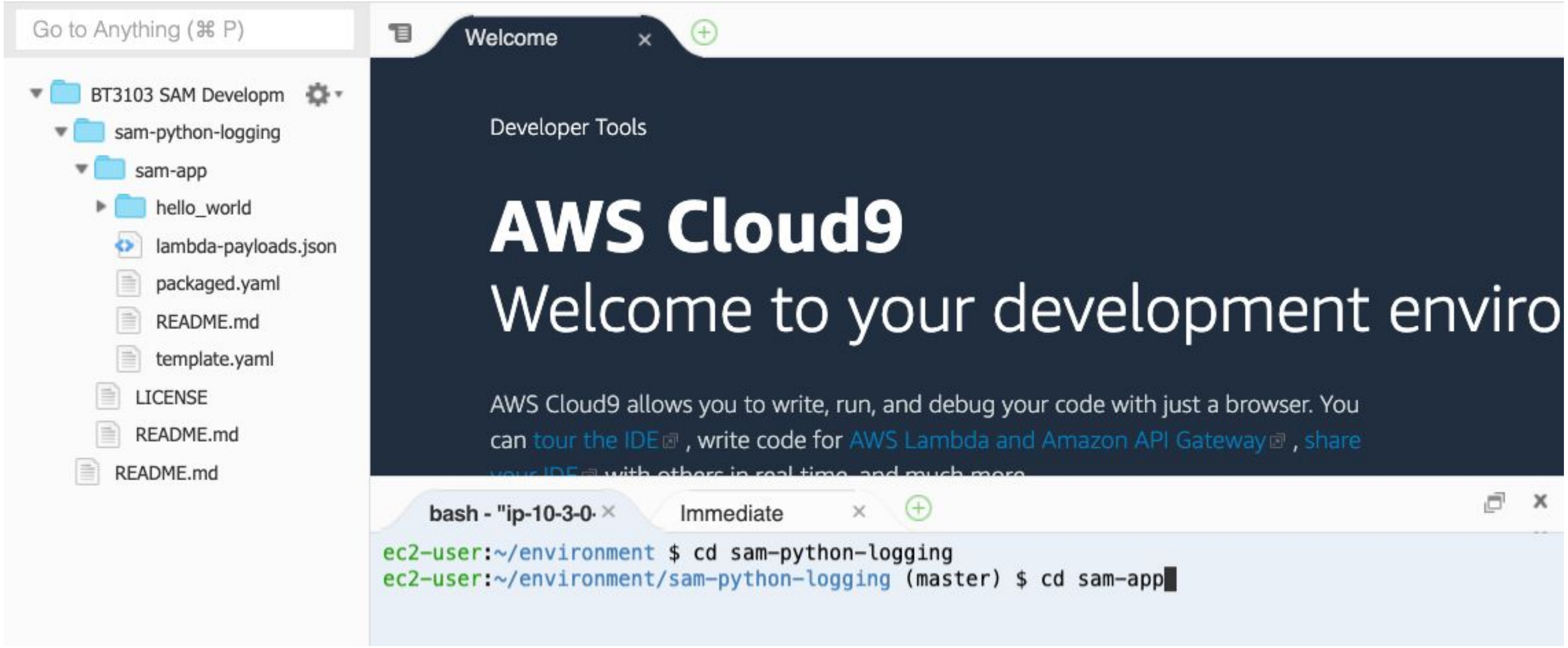


AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share

```
ec2-user:~/environment $ git clone https://github.com/scboesch/sam-python-logging
Cloning into 'sam-python-logging'...
remote: Enumerating objects: 82, done.
remote: Counting objects: 100% (82/82), done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 82 (delta 45), reused 60 (delta 26), pack-reused 0
Unpacking objects: 100% (82/82), done.
ec2-user:~/environment $ cd sam-python-logging
```

Change directory into the sam-app directory.

- cd sam-app



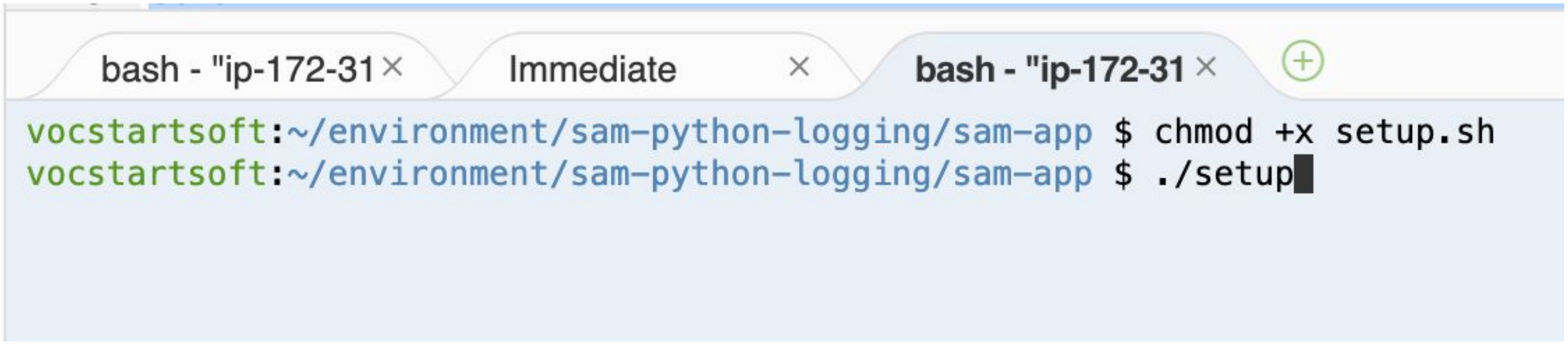
AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share

```
ec2-user:~/environment $ cd sam-python-logging
ec2-user:~/environment/sam-python-logging (master) $ cd sam-app
```

Update SAM CLI version on Cloud 9

In the cloud9 terminal run the Cloud9 setup script

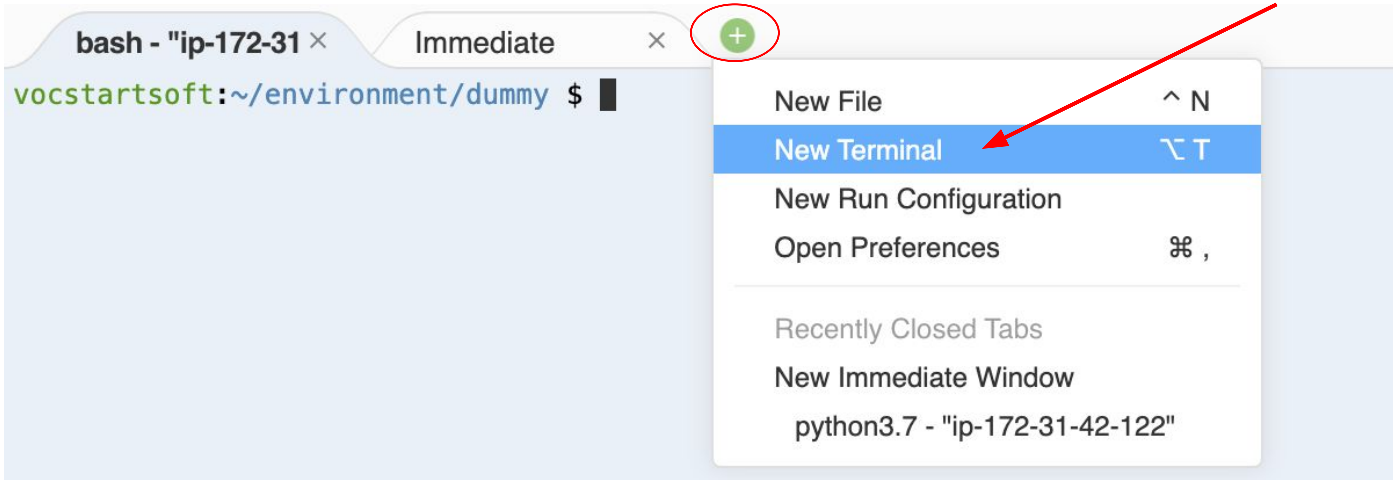
- **chmod +x c9-setup.sh**
- **./c9-setup.sh**



```
vocstartsoft:~/environment/sam-python-logging/sam-app $ chmod +x setup.sh
vocstartsoft:~/environment/sam-python-logging/sam-app $ ./setup
```

Update SAM CLI version on Cloud 9

After the script runs, switch to a new terminal so that the changes can take effect.



```
vocstartsoft:~/environment/dummy $
```

In the new terminal run -

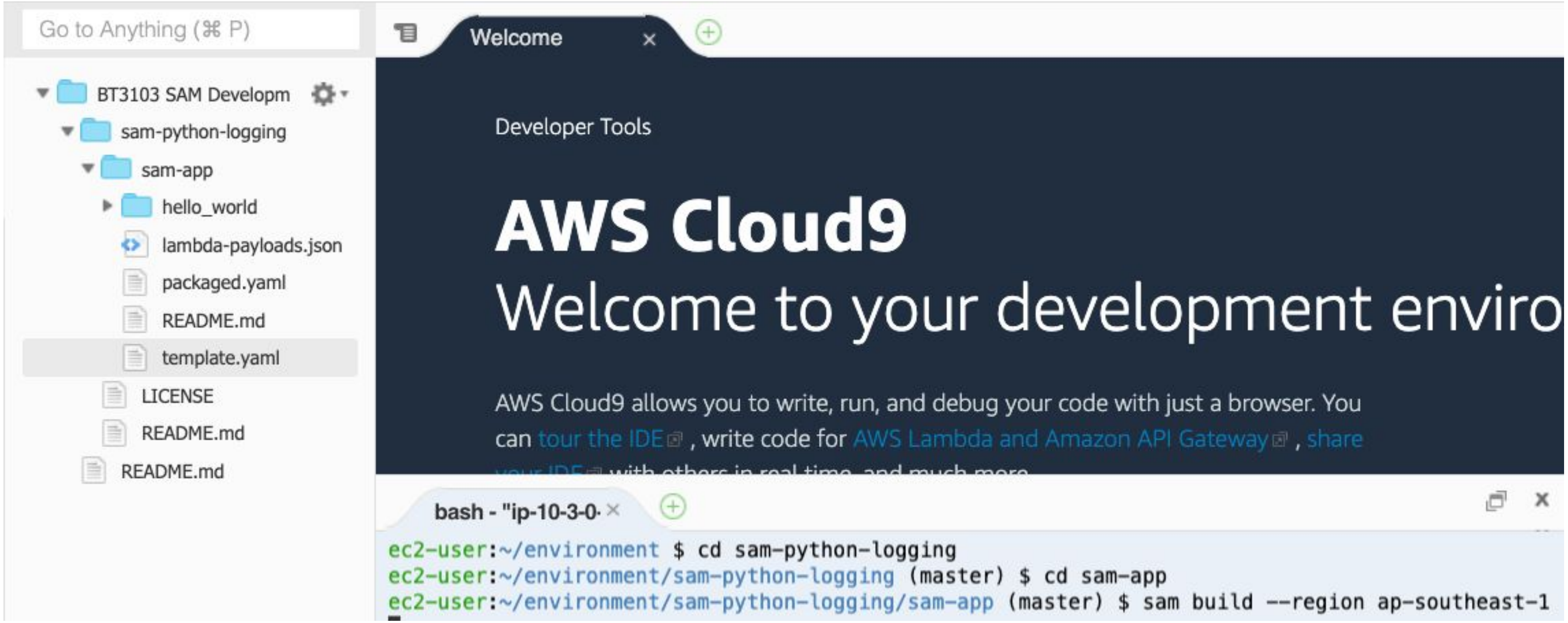
- sam --version

Sam version installed would be v0.22.0

Continue the rest of the slides with this new terminal.

Build the application for the Singapore region ap-southeast-1

- sam build --region ap-southeast-1



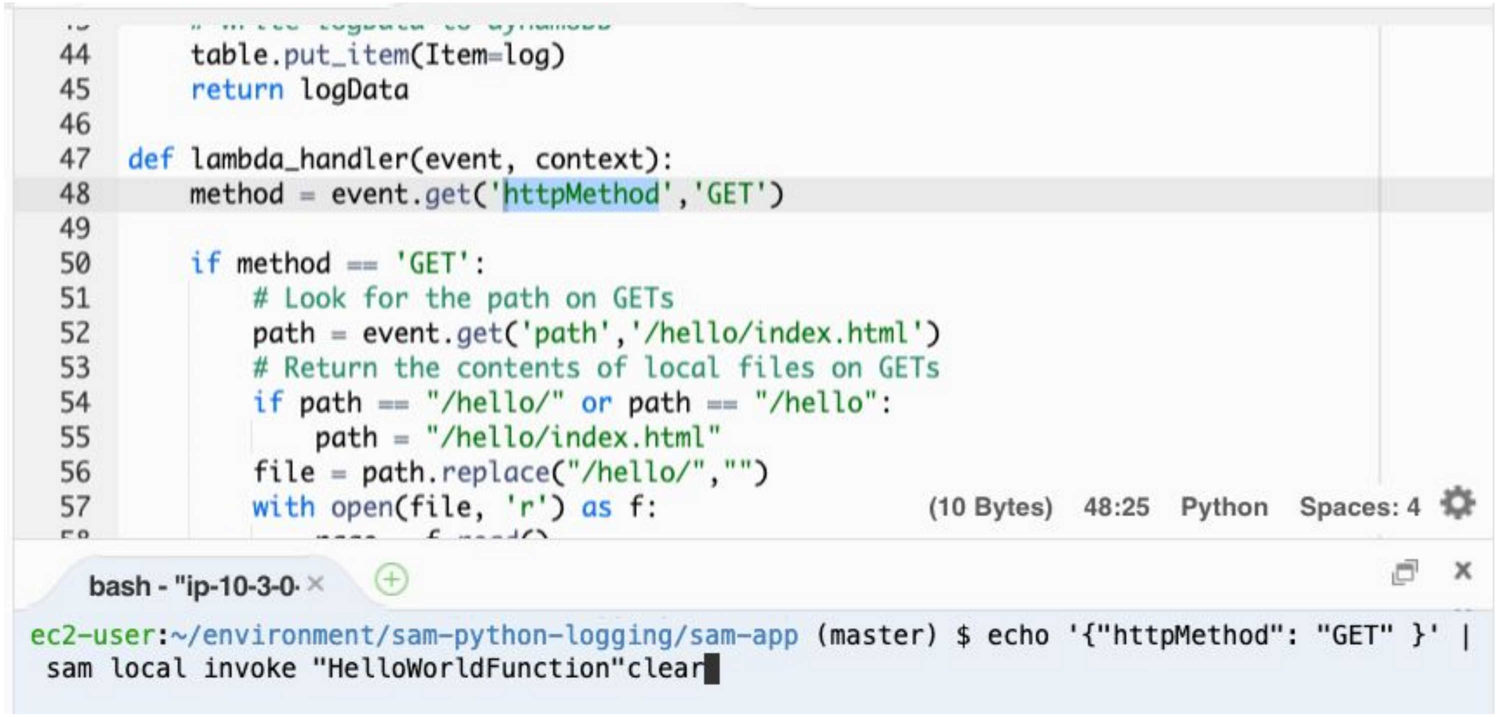
Welcome to your development enviro

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.

```
ec2-user:~/environment $ cd sam-python-logging
ec2-user:~/environment/sam-python-logging (master) $ cd sam-app
ec2-user:~/environment/sam-python-logging/sam-app (master) $ sam build --region ap-southeast-1
```

Locally invoke the HelloWorldFunction. You can pass-in example events to the sam local invoke function.

- echo '{"httpMethod": "GET" }' | sam local invoke "HelloWorldFunction"

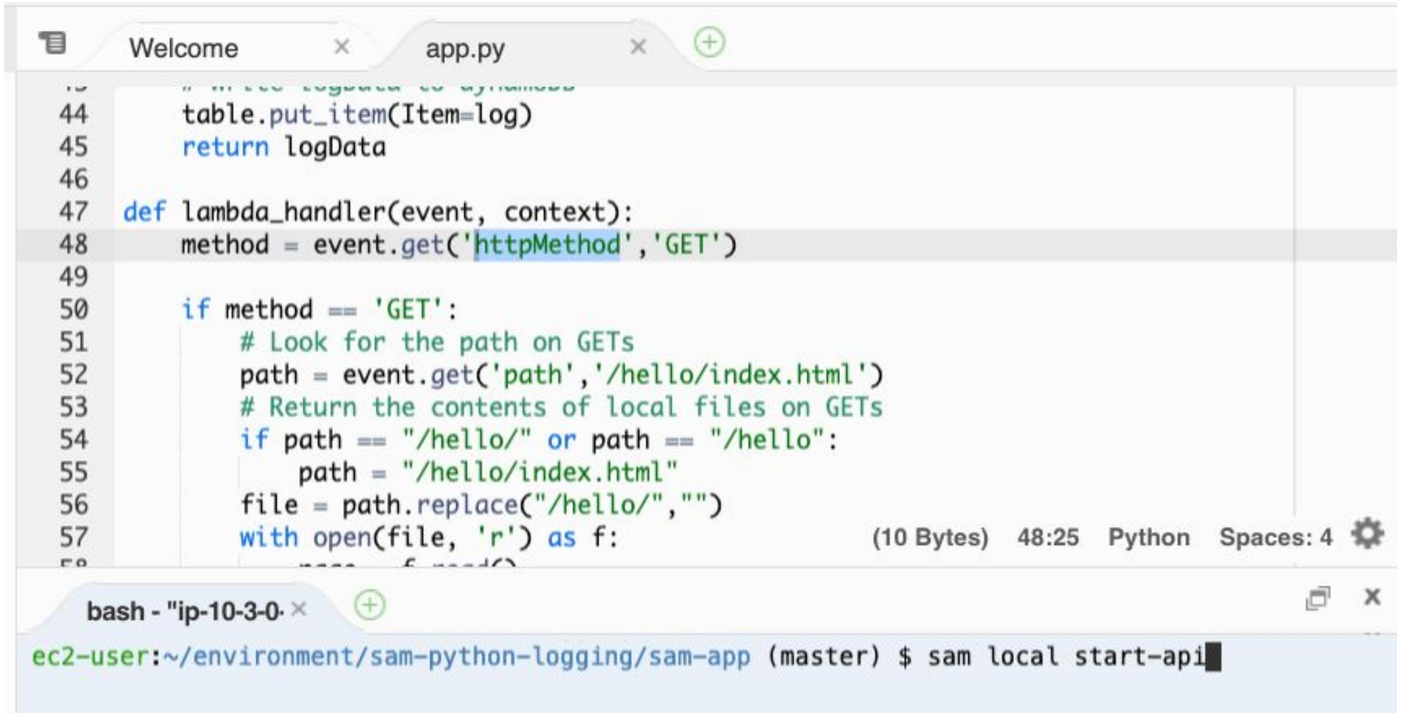


```
    table.put_item(Item=log)
    return logData

def lambda_handler(event, context):
    method = event.get('httpMethod','GET')

    if method == 'GET':
        # Look for the path on GETs
        path = event.get('path','/hello/index.html')
        # Return the contents of local files on GETs
        if path == "/hello/" or path == "/hello":
            path = "/hello/index.html"
        file = path.replace("/hello/","")
        with open(file, 'r') as f:
```

```
ec2-user:~/environment/sam-python-logging/sam-app (master) $ echo '{"httpMethod": "GET" }' |
 sam local invoke "HelloWorldFunction"clear
```

Start the API locally

- sam local start-api

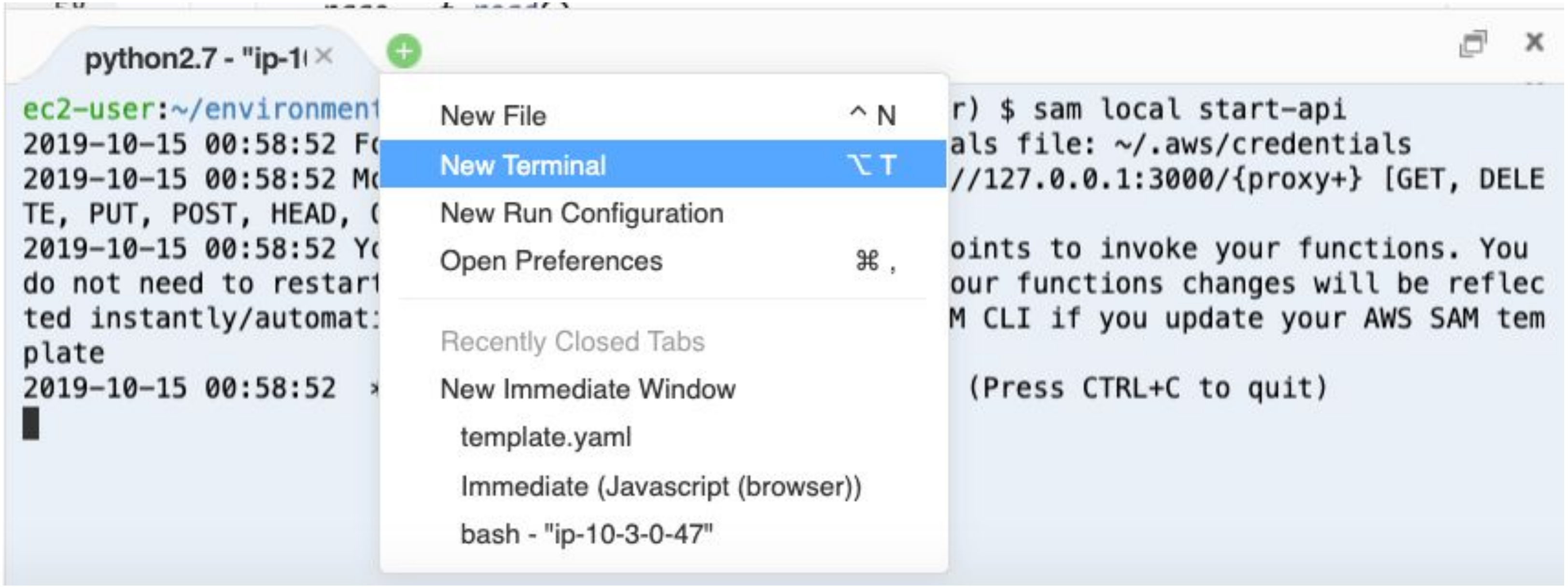


```python
    table.put_item(Item=log)
    return logData

def lambda_handler(event, context):
    method = event.get('httpMethod','GET')

    if method == 'GET':
        # Look for the path on GETs
        path = event.get('path','/hello/index.html')
        # Return the contents of local files on GETs
        if path == "/hello/" or path == "/hello":
            path = "/hello/index.html"
        file = path.replace("/hello/","")
        with open(file, 'r') as f:
```

(10 Bytes) 48:25 Python Spaces: 4

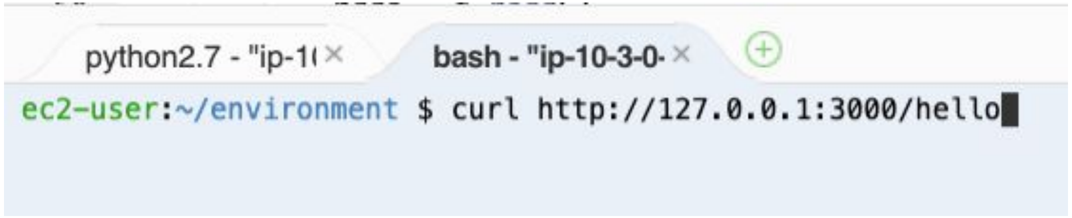
```
ec2-user:~/environment/sam-python-logging/sam-app (master) $ sam local start-api
```

Create a new terminal and run curl from the new terminal.



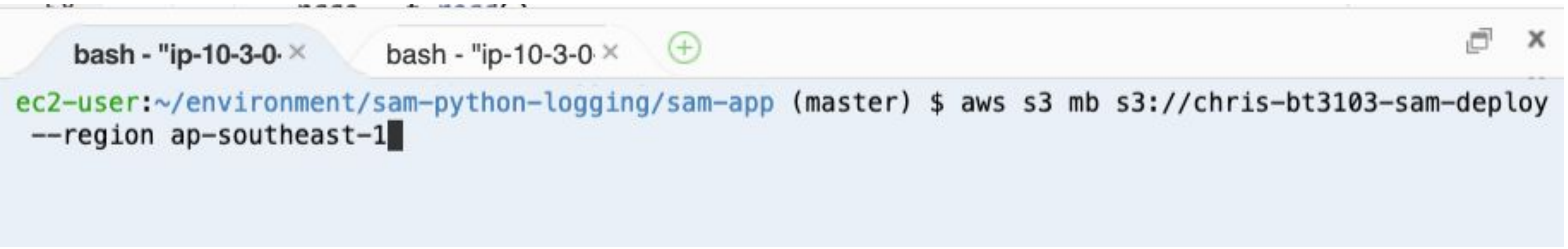
```
ec2-user:~/environment                              r) $ sam local start-api
2019-10-15 00:58:52 F(                              als file: ~/.aws/credentials
2019-10-15 00:58:52 M(                              //127.0.0.1:3000/{proxy+} [GET, DELE
TE, PUT, POST, HEAD, (
2019-10-15 00:58:52 Y(                              oints to invoke your functions. You
do not need to restart                              our functions changes will be reflec
ted instantly/automat:                              M CLI if you update your AWS SAM tem
plate
2019-10-15 00:58:52  *                               (Press CTRL+C to quit)
```

- curl http://127.0.0.1:3000/hello



```
ec2-user:~/environment $ curl http://127.0.0.1:3000/hello
```

Go back to the first terminal and stop the local server by pressing control-c.
Make a new S3 bucket with a unique name to deploy your app. You will need to change the bucket name.



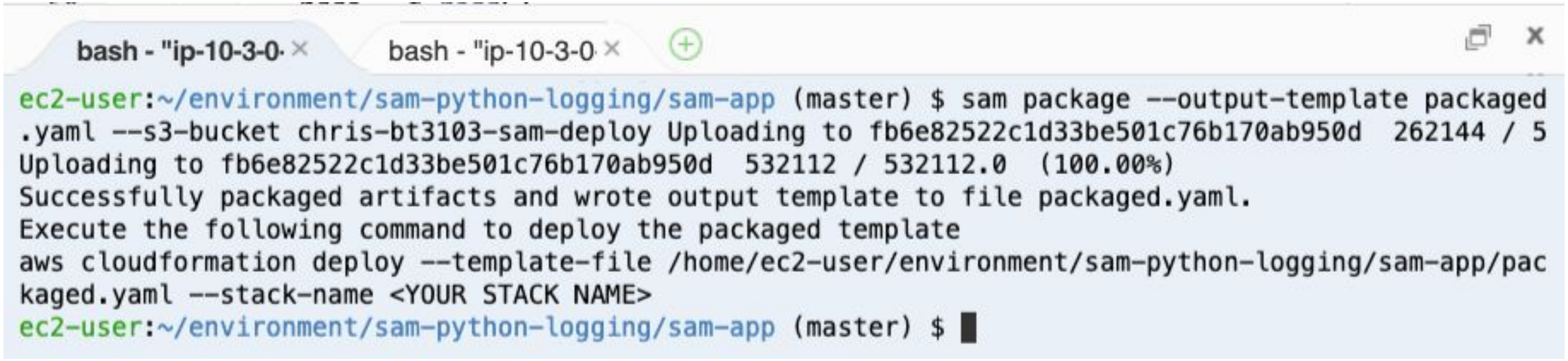
```
ec2-user:~/environment/sam-python-logging/sam-app (master) $ aws s3 mb s3://chris-bt3103-sam-deploy
 --region ap-southeast-1
```

- aws s3 mb s3://<name-your-new-s3-bucket> --region ap-southeast-1

Aws s3 mb command creates a new bucket with the specified name. Specify a unique name for your bucket.

Run the package command to generate a packaged.yaml file

- sam package --output-template packaged.yaml --s3-bucket <s3-bucket-name>

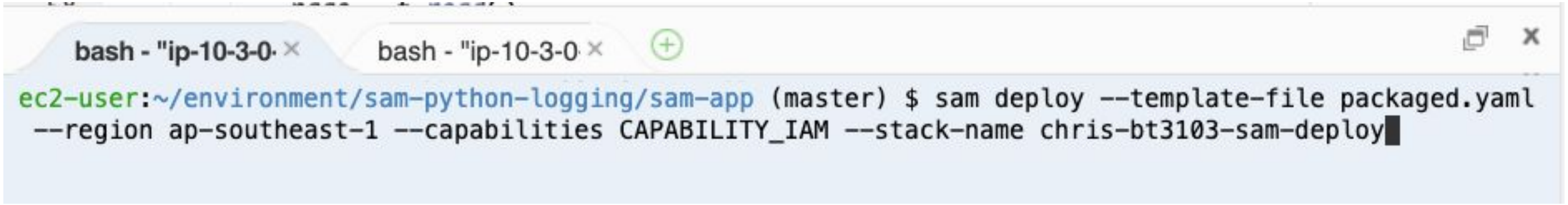


bash - "ip-10-3-0· bash - "ip-10-3-0·

```
ec2-user:~/environment/sam-python-logging/sam-app (master) $ sam package --output-template packaged
.yaml --s3-bucket chris-bt3103-sam-deploy Uploading to fb6e82522c1d33be501c76b170ab950d  262144 / 5
Uploading to fb6e82522c1d33be501c76b170ab950d  532112 / 532112.0  (100.00%)
Successfully packaged artifacts and wrote output template to file packaged.yaml.
Execute the following command to deploy the packaged template
aws cloudformation deploy --template-file /home/ec2-user/environment/sam-python-logging/sam-app/pac
kaged.yaml --stack-name <YOUR STACK NAME>
ec2-user:~/environment/sam-python-logging/sam-app (master) $ █
```

Deploy the cloudformation stack

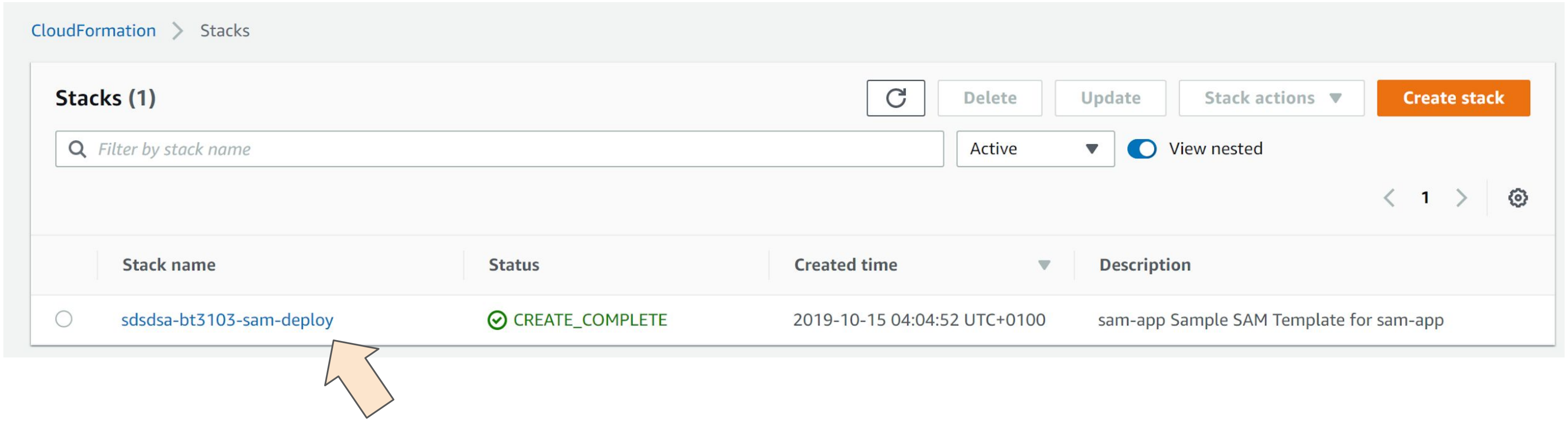
- sam deploy --template-file packaged.yaml --region ap-southeast-1 --capabilities CAPABILITY_IAM --stack-name <cloudformation-stack-name-tobecreated>



```
ec2-user:~/environment/sam-python-logging/sam-app (master) $ sam deploy --template-file packaged.yaml
 --region ap-southeast-1 --capabilities CAPABILITY_IAM --stack-name chris-bt3103-sam-deploy
```

Viewing the results - Cloudformation

- Head over to Cloudformation



CloudFormation > Stacks

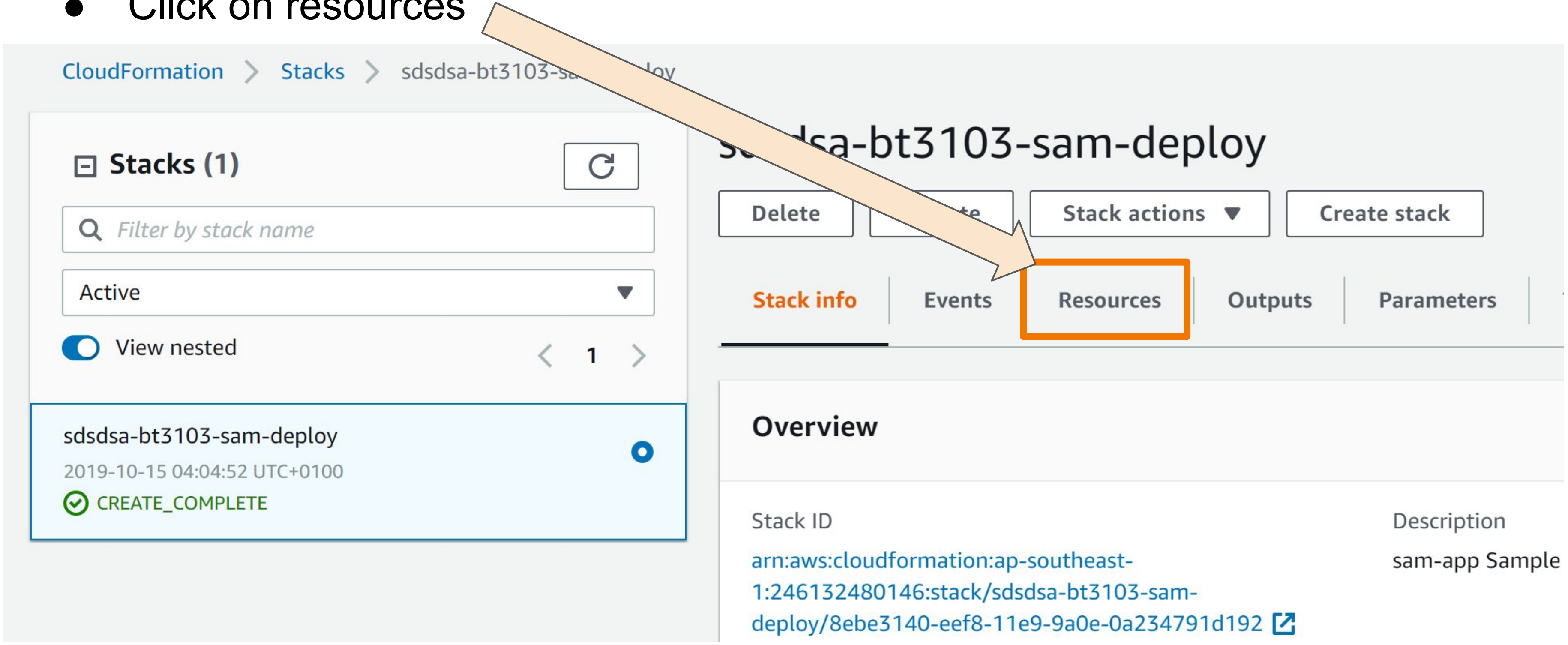
Stacks (1)

Stack name	Status	Created time	Description
sdsdsa-bt3103-sam-deploy	CREATE_COMPLETE	2019-10-15 04:04:52 UTC+0100	sam-app Sample SAM Template for sam-app

- Click on the stack you have created

Viewing the results - Cloudformation

- Click on resources



sdsdsa-bt3103-sam-deploy
2019-10-15 04:04:52 UTC+0100
CREATE_COMPLETE

sdsa-bt3103-sam-deploy

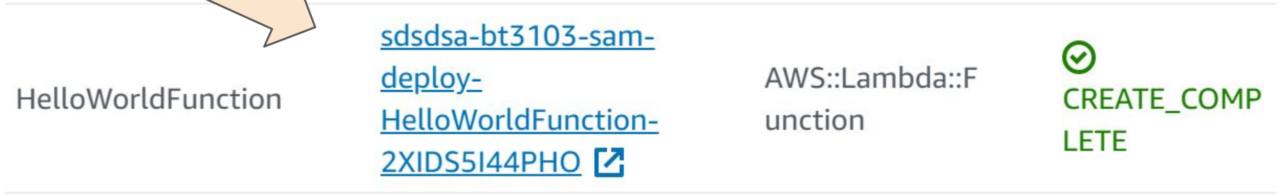
Overview

Stack ID
arn:aws:cloudformation:ap-southeast-1:246132480146:stack/sdsdsa-bt3103-sam-deploy/8ebe3140-eef8-11e9-9a0e-0a234791d192

Description
sam-app Sample

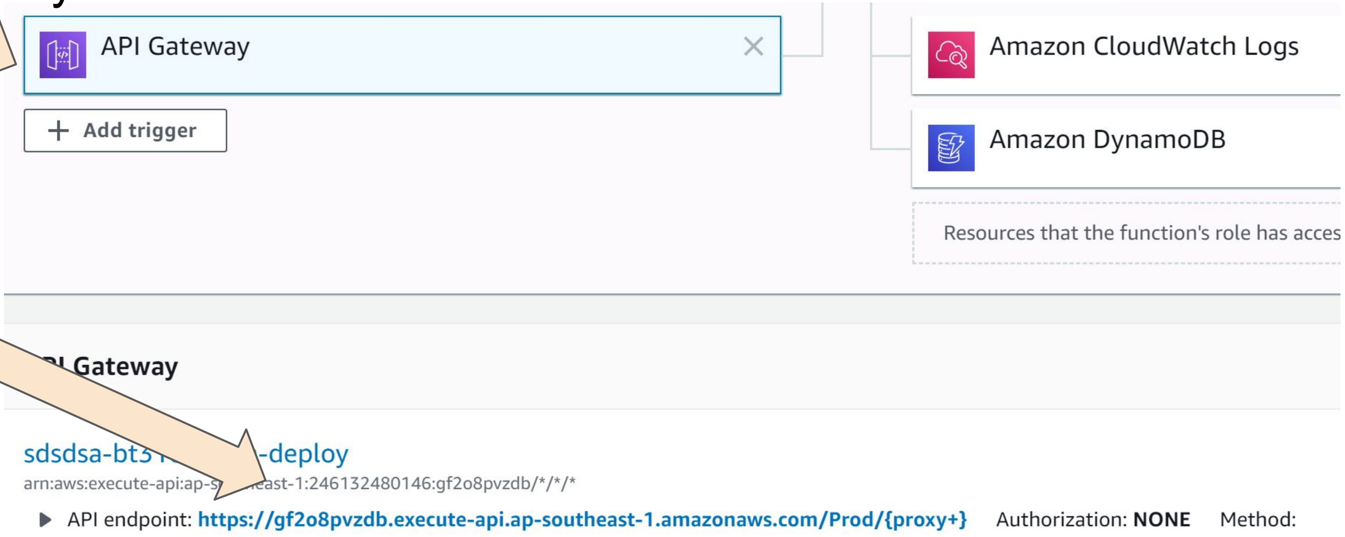
Viewing the results - Lambda Function

- Click on HelloWorldFunction



HelloWorldFunction | sdsdsa-bt3103-sam-deploy-HelloWorldFunction-2XIDS5I44PHO | AWS::Lambda::Function | CREATE_COMPLETE

- Click on API Gateway
- And then API endpoint



API Gateway

Add trigger

Amazon CloudWatch Logs

Amazon DynamoDB

Resources that the function's role has acces

PI Gateway

sdsdsa-bt3 -deploy

arn:aws:execute-api:ap-s east-1:246132480146:gf2o8pvzdb/*/*/*

API endpoint: https://gf2o8pvzdb.execute-api.ap-southeast-1.amazonaws.com/Prod/{proxy+} Authorization: NONE Method:

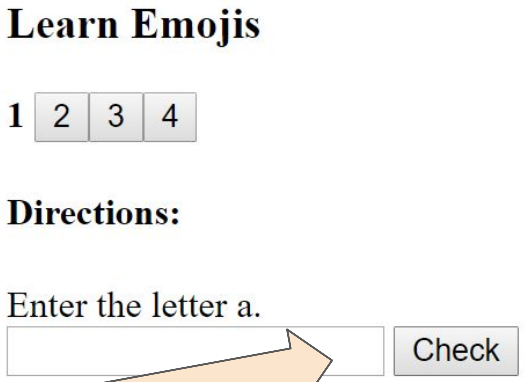
Viewing the results - Lambda Function

- Change the last part of the url to /hello/

https://……………...execute-api.ap-southeast-1.amazonaws.com/Prod/**hello/**

- You should see this

- Complete some activities


Learn Emojis

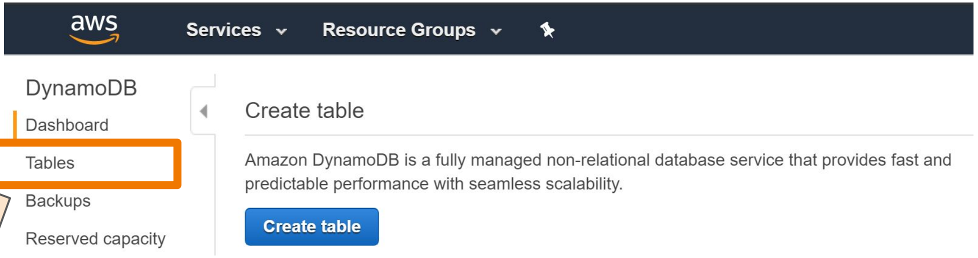
1 2 3 4

Directions:

Enter the letter a.

Viewing the results - DynamoDB

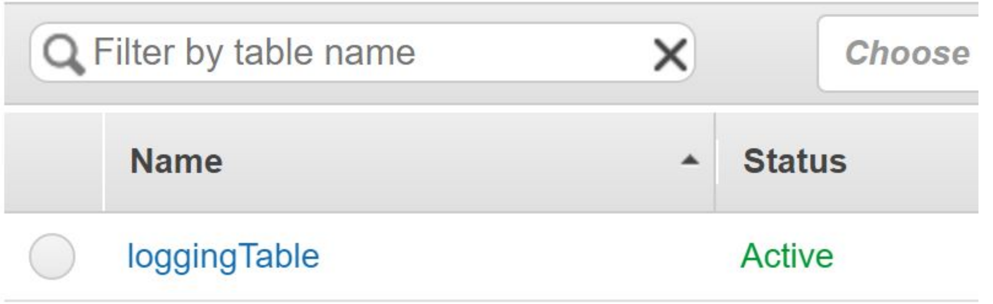
- Head over to DynamoDB service (Singapore region)



Amazon DynamoDB is a fully managed non-relational database service that provides fast and predictable performance with seamless scalability.

- Click on tables

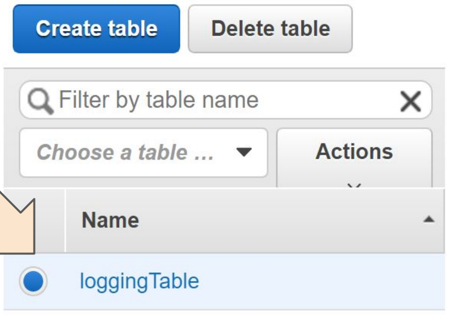
- Click on loggingTable

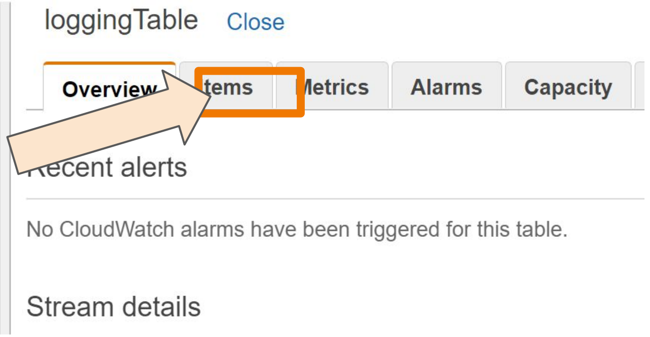


	Name	Status
	loggingTable	Active

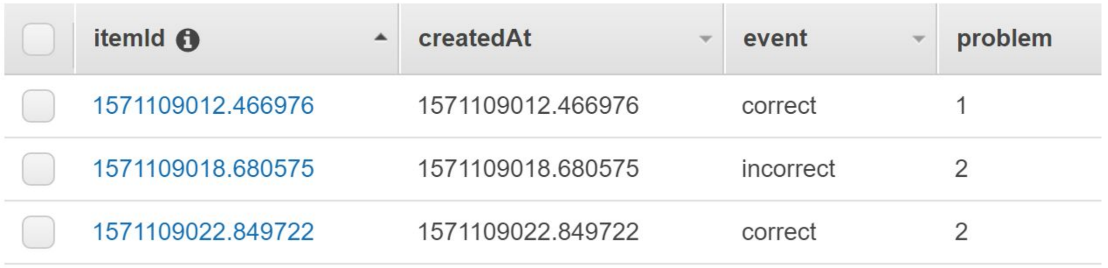
Viewing the results - DynamoDB

- Click on loggingTable
- Then, click on Items
- Your interaction logs should be there





No CloudWatch alarms have been triggered for this table.



itemId	createdAt	event	problem
1571109012.466976	1571109012.466976	correct	1
1571109018.680575	1571109018.680575	incorrect	2
1571109022.849722	1571109022.849722	correct	2