**COMPX341-22A Assignment 4**

**Quality Assurance via CI/CD on the cloud**

**Prepared by: Kevin Han 1521885**

**Table of Content**

Contents

[1. Overview 2](#_Toc106377425)

[2. Task 1: Preparing your Cloud9 IDE 2](#_Toc106377426)

[3. Task 2: Initializing AWS Serverless Architecture Model 6](#_Toc106377427)

[4. Task 3: Deploy Locally 9](#_Toc106377428)

[5. Task 4: Configuring your repository 11](#_Toc106377429)

[6. Task 5: Implementing a CI/CD pipeline 13](#_Toc106377430)

[7. Task 6: Implementing Functionality and Testing 16](#_Toc106377431)

[8. Conclusion 21](#_Toc106377432)

# Overview

The purpose of the report is to document the progress of the steps to understand the importance of AWS and DevOps. The outcome of is to be able to utilise DevOps services on the cloud and apply DevOps and testing processes to deploy applications on the cloud and automate a continuous deployment pipeline. Also show that errors will not be push to repository stored on the cloud/system.

We will be using the AWS (<https://awsacademy.instructure.com/>) to demonstrate my understanding of the task provided.

# Task 1: Preparing your Cloud9 IDE

The task is to initialise the environment for the Cloud9 IDE, to have access the IDE from any logged-in AWS console.

Step 1: Search for Cloud9 in the AWS Console as shown in Fig 1.

Fig

Graphical user interface, text, application

Description automatically generated

Step 2: Click on the create environment as shown in Fig 2.

Fig

Graphical user interface

Description automatically generated

Step 3: Set-up the environment name based on the assignment specification. The name should be like “your\_name-studentID-assignment4”. It should look like Fig 3.

Fig

Graphical user interface, text, application, email

Description automatically generated

Step 4: The application will not be using any special services, so the setting can be left as default. It should look like Fig 4-5.

Fig

Graphical user interface, text, application

Description automatically generated

Fig

Graphical user interface, text, application

Description automatically generated

Step 5: Click on the create environment button to create the environment.

Fig

Graphical user interface, text, application

Description automatically generated

After clicking the button, it will show the screen based on the Fig 7. This shows that the AWS environment is currently being created.

Fig

Graphical user interface, text, application

Description automatically generated

Open the environment and type “aws sts get-caller-identity” in the console. This is to ensure that your account is signed in. An example of is shown in Fig 8.

Fig

Text

Description automatically generated

# Task 2: Initializing AWS Serverless Architecture Model

The task is to initialise the AWS Serverless Architecture Model to the latest version within the Cloud9 IDE. We will be using a preset template provided to initalise the “AWS Hello World Example”.

Step 1: Run the following commands.

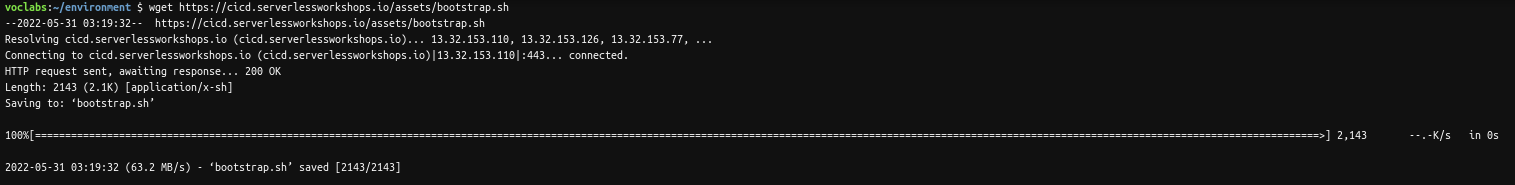
wget https://cicd.serverlessworkshops.io/assets/bootstrap.sh

chmod +x bootstrap.sh

./bootstrap.sh

The following command is to get the latest version, then allowing the program to be executable, then run the bootstrap script. It should look like Fig 9-10 when you run the command.

Fig



Fig

Text

Description automatically generated

The following command “sam –version” shows the current version of the Serverless Architecture Model. At this current moment, the latest version is show in the Fig 11.

Fig



Step 2: Initialize the Serverless Architecture Model by running the command “sam init”.

This is to initialize the Serverless Architecture Model and we want to use the pre-set hello world template.

Follow the choice as shown in the Fig 12-13.

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

After initializing the Serverless Architecture Model, you should see the directory of the project on the menu at the left side of the screen as shown in Fig 14.

Fig

Text

Description automatically generated

# Task 3: Deploy Locally

The task is to be able to run the project locally. To do this we need to install some dependency.

Step 1: Install the prebuilt OpenJDK packages, update the configs for the java and javac.

Run the following commands:

sudo yum -y install java-1.8.0-openjdk-devel

sudo update-alternatives --config java

sudo update-alternatives --config javac

The Fig 15-16 shows the outcome as you enter the commands.

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Step 2: Install maven, it is used to by the project to build.

Run the following commands:

sudo wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo

sudo sed -i s/\$releasever/7/g /etc/yum.repos.d/epel-apache-maven.repo

sudo yum install -y apache-maven

The Fig 17-18 shows what it is expected when you run the commands to install maven.

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Step 3: We can now build the project from the template file.

Ensure that you are in the right directory, run the command “sam build”. If done correctly, you should see that the console will output “Build Succeeded” as shown in the Fig 19. This shows the project has been successfully built.

Fig

Text

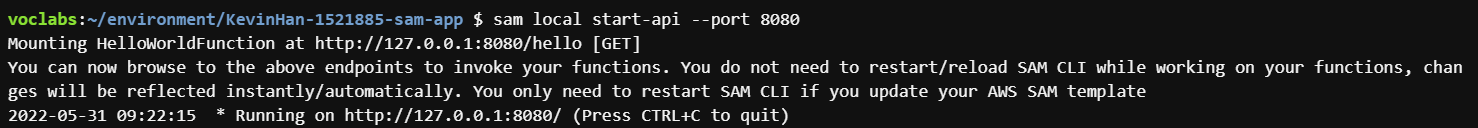
Description automatically generated

Step 4: To deploy the project locally after the successful build.

To deploy the project, ensure that you are currently in yourname-studentid-sam-app folder. Then run the command “sam local start-api --port 8080” to deploy the project.

The Fig 20 shows the project has deploy.

Fig



To test whether the project is working locally, we can send “curl http://localhost:8080/hello” in another terminal. The output should be the same as the Fig 21.

Fig



The Fig 22 shows the request made by the curl command above.

Fig

Text

Description automatically generated

# Task 4: Configuring your repository

The task is to have our latest version of deployed project to be stored in a cloud repository. For this assignment, we are storing the repository in aws’s git service.

Step 1: Creating the git repository

Run the command ‘aws codecommit create-repository --repository-name “yournamesamapp-repo”’ The output of the command is shown in Fig 23.

Fig

Text

Description automatically generated

Step 2: After successfully creating the repo, we need to configure the git repo with your details. To allow us to know who the commits are made by. Run the command shown in Fig 24.

Fig



Step 3: We need to config git to ignore some files. Create a .gitignore file, if it is not showing, enable hidden files. We do not want to store the built applications, so add the following as show in the fig 25.

Fig

A screenshot of a computer

Description automatically generated with medium confidence

Step 4: We can now population our repository with the project files. To do this we need to run the following commands “git init”, “git add .” and “git commit -m “Initial commit””.

Git init – to initialise a new local repository which store all the object and ref used in the project and creates a project’s history.

Git add . – add the updates made to the project

Git commit – it captures the snapshot of the current stage changes of the project.

When you run the commands, it should look similar to the Fig 26-27.

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Step 5: Now we can add the remote origin that we created in step 1 and push the repository to the “cloneUrlHttp”. Follow the commands show in Fig 28-29.

Fig



Fig

Text

Description automatically generated

After pushing the repository, you should be able to see your populated repository in the AWS codecommit console. Your populated repository should look like Fig 30.

Fig

Graphical user interface, text, application, email

Description automatically generated

# Task 5: Implementing a CI/CD pipeline

The task is to create a bash script which automates the testing, building, committing, and pushing. If the tests are successful, it will attempt to build. If the build is successful, it will commit and push to the AWS Code Commit and ask the user if they want to deploy the project. If the build fails, it should not commit and push to AWS Code Commit. If the tests fails, it should not build, commit, and push to AWS Code Commit.

Fig

Text

Description automatically generated

I have setup the bash script as shown in Fig 31. The bash script first check if it can run the test successfully, then check if can build successfully. If it is build successful, it will ask the user for a commit message before committing and pushing the AWS Code Commit. Then ask the user whether they want to deploy locally. The user are given 2 choices Y or N. If other input are given, it will give an error and ask the user again. Error outputs are shown if the test or build are to fail. I decided to developed this way based on the assignment specification. The following Fig 32-36 shows my pipeline script running.

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Fig

Graphical user interface, application

Description automatically generated

Fig

Text

Description automatically generated with medium confidence

# Task 6: Implementing Functionality and Testing

The task is to make some changes to the java application and testing the pipeline and application. The changes are to allow the program to access the S3 bucket and retrieve the quotes stored and randomly display a quote back when a curl is made. Testing must ensure the quote exists in the quotes.txt retrieve from the bucket.

Fig 37 shows the test failed when we made a change to the hello world message in the App.java.

Fig

Text

Description automatically generated

The following Fig 38-39 show the bucket that has been created and the storing the quotes.txt file in the bucket.

Fig

Graphical user interface, text, application

Description automatically generated

Fig

Graphical user interface, text, application, email

Description automatically generated

Fig 40 show the imports of the java and aws service that I will be using for the changes made in this task.

Fig 41-42 shows the dependency needed for the App.java to retrieve the contents of the buckets. We are getting the particular dependency instead of the whole Maven SDK Modules, so it doesn't take a long time building the application. The dependency are added to the pom.xml.

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Fig

Text

Description automatically generated

Fig 43 show the AppTest.java that has been modified to check whether the quote parse contains within the quotes.txt file.

Fig

Text

Description automatically generated

Fig 44 show the code that parse in from the bucket. Make sure that the region of the bucket is based on the region you created your bucket. Since the SAM runs on the cloud, it is not able to read and write 2 files. So, I decided to convert s3is input stream to a string as shown in the Fig 44. I added a file write which will write the content of the string to a file which is used by the AppTest.java to check whether the quote exists.

Fig

Text

Description automatically generated

Fig 45 shows the string receive from the bucket being separated by the line separator and stored in an array list called quoteList.

Fig 46, the output has been modified to output the quote instead of the original hello message.

Fig



Fig

Text

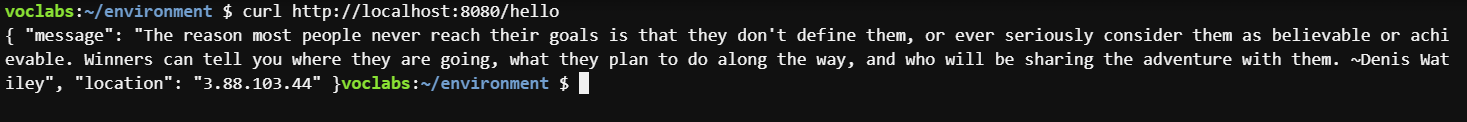
Description automatically generated

Fig

Text

Description automatically generated with medium confidence

Fig



Fig

Text

Description automatically generated

Fig 47-50 shows the test being conducted after the changes has been made to the assignment brief. We can see the message response with a quote from the quotes.txt from the bucket list. For the mvn test, we are testing whether the quote exist within the quote.txt, the http status code is 200.

# Conclusion

The purpose of the project is to show our understanding of implementation of functionality and testing of the changes made. Allowing automate scripts allow us as software engineers to efficiently test and push the latest working deploy project to the repository.