Creating&Managing a Serverless Project(Python app+pipeline+lamda+Api)in AWS CodeStar

We use AWS CodeStar to create a project that uses the AWS Serverless Application Model (AWS SAM) to create and manage AWS resources for a web service hosted in AWS Lambda.

AWS CodeStar uses AWS SAM, which relies on AWS CloudFormation, to provide a simplified way of creating and managing supported AWS resources, including Amazon API Gateway APIs, AWS Lambda functions, and Amazon DynamoDB tables. (This project does not use any Amazon DynamoDB tables.)

Prerequisite: Complete the steps in Setting Up AWS CodeStar.
Note

Your AWS account might be charged for costs related to this tutorial, including costs for AWS services used by AWS CodeStar.

Topics

Overview

In this tutorial, you:

1. Use AWS CodeStar to create a project that uses AWS SAM to build and deploy a Python-based web service. This web service is hosted in AWS Lambda and can be accessed through Amazon API Gateway.
2. Explore the project's main resources, which include:
   - The AWS CodeCommit repository where the project's source code is stored. This source code includes the web service's logic and defines related AWS resources.
   - The AWS CodePipeline pipeline that automates the building of the source code. This pipeline uses AWS SAM to create and deploy a function to AWS Lambda, create a related API in Amazon API Gateway, and connect the API to the function.
   - The function that is deployed to AWS Lambda.
   - The API that is created in Amazon API Gateway.
3. Test the web service to confirm that AWS CodeStar built and deployed the web service as expected.
4. Set up your local workstation to work with the project's source code.
5. Change the project's source code using your local workstation. When you add a function to the project and then push your changes to the source code, AWS CodeStar rebuilds and redeploys the web service.
6. Test the web service again to confirm that AWS CodeStar rebuilt and redeployed as expected.
7. Write a unit test using your local workstation to replace some of your manual testing with an automated test. When you push the unit test, AWS CodeStar rebuilds and redeploys the web service and runs the unit test.
8. View the results of the unit tests.
9. Clean up the project. This step helps you avoid charges to your AWS account for costs related to this tutorial.

Step 1: Create the Project

In this step, you use the AWS CodeStar console to create a project.

1. Sign in to the AWS Management Console and open the AWS CodeStar console, at https://console.aws.amazon.com/codestar/.
   Note

   You must sign in to the AWS Managemen[illegible]ciated with the IAM user you created or identified in Setting Up AWS CodeStar. This user must have the **`AWSCodeStarFullAccess`** managed policy attached.
2. Choose the AWS Region where you want to create the project and its resources.
3. Choose **Create a new project**. If **Create a new project** is not displayed, choose **Start a new project**.
4. On the **Choose a project template** page:
   - For **Application category**, select **Web service**.
   - For **Programming languages**, select **Python**.
   - For **AWS services**, select **AWS Lambda**.
5. Choose the box that contains your selections.
6. For **Project name**, enter a name for the project (for example, **`My SAM Project`**). If you use a name different from the example, be sure to use it throughout this tutorial.

For **Project ID**, AWS CodeStar chooses a related identifier for this project (for example, **my-sam-project**). If you see a different project ID, be sure to use it throughout this tutorial.

Leave **AWS CodeCommit** selected, and do not change the **Repository name** value.

7. Choose **Next**.
8. Leave **AWS CodeStar would like permission to administer AWS resources on your behalf** selected, and then choose **Create Project**.

   If this is your first time using AWS CodeStar in this AWS Region, for **Display Name** and **Email**, enter the display name and email address you want AWS CodeStar to use for your IAM user. Choose **Next**.
9. On **Choose how you want to edit your project code**, choose **Skip**. You set up your local workstation to edit the project's code in a later step.
10. Wait while AWS CodeStar creates the project. This might take several minutes. Do not continue until you see **Welcome to My SAM Project!**.

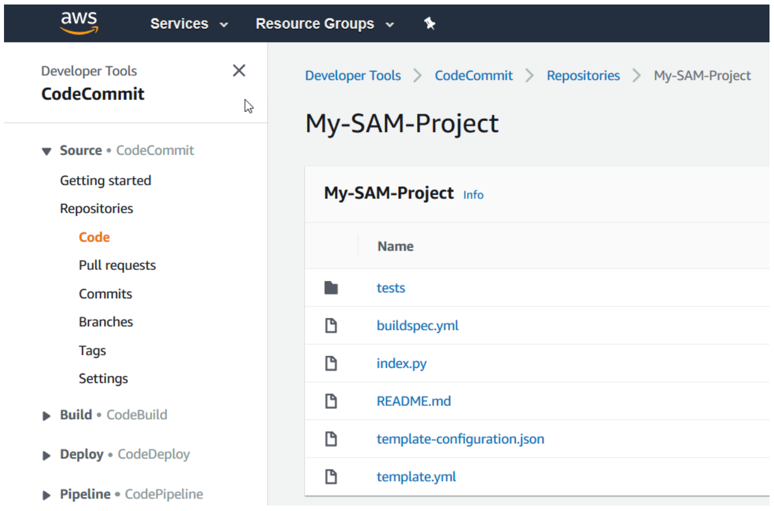
Step 2: Explore Project Resources

In this step, you explore four of the project's AWS resources to understand how the project works:

- The AWS CodeCommit repository where the project's source code is stored. AWS CodeStar gives the repository the name **my-sam-proj ect**, where **my-sam-project** is the name of the project.
- The AWS CodePipeline pipeline that uses CodeBuild and AWS SAM to automate building and deploying the web service's Lambda function and API in API Gateway. AWS CodeStar gives the pipeline the name **my-sam-project--Pipeline**, where **my-sam-project** is the ID of the project.
- The Lambda function that contains the logic of the web service. AWS CodeStar gives the function the name **awscodestar-my-sam-proj ect-lambda-HelloWorld-*`RANDOM_ID`***, where:
  - **my-sam-project** is the ID of the project.
  - **HelloWorld** is the function ID as specified in the `template.yaml` file in the AWS CodeCommit repository. You explore this file later.
  - *`RANDOM_ID`* is a random ID that AWS SAM assigns to the function to help ensure uniqueness.
- The API in API Gateway that makes it easier to call the Lambda function. AWS CodeStar gives the API the name **awscodestar-my-sam-project--lambda**, where **my-sam-project** is the ID of the project.

To explore the source code repository in CodeCommit

1. With your project open in the AWS CodeStar console, on the side navigation bar, choose **Code**.
2. In the CodeCommit console, on the **Code** page, the source code files for the project are displayed:
   - `buildspec.yml`, which CodePipeline instructs CodeBuild to use during the build phase, to package the web service using AWS SAM.
   - `index.py`, which contains the logic for the Lambda function. This function simply outputs the string `Hello World` and a timestamp, in ISO format.
   - `README.md`, which contains general information about the repository.
   - `template-configuation.json`, which contains the project ARN with placeholders used for tagging resources with the project ID
   - `template.yml`, which AWS SAM uses to package the web service and create the API in API Gateway.



My-SAM-Project

My-SAM-Project Info

Name
tests
buildspec.yml
index.py
README.md
template-configuration.json
template.yml

To explore the pipeline in CodePipeline

1. To view information about the pipeline, with your project open in the AWS CodeStar console, on the side navigation bar, choose **Dashbo ard**. On the **Continuous deployment** tile, you see the pipeline contains:

- A **Source** stage for getting the source code from CodeCommit.
- A **Build** stage for building the source code with CodeBuild.
- A **Deploy** stage for deploying the built source code and AWS resources with AWS SAM.

2. To view more information about the pipeline, on the **Continuous deployment** tile, choose the **CodePipeline details** link or, on the side navigation bar, choose **Pipeline** to open the pipeline in the CodePipeline console.

To explore AWS service resources on the Project page

1. Open your project in the AWS CodeStar console and from the navigation bar, choose **Project**.
2. Review the **Project Details** and **Project Resources** lists.

To explore the function in Lambda

1. With your project open in the AWS CodeStar console, on the side navigation bar, choose **Project**.
2. In **Project Resources**, in the **ARN** column, choose the link for the Lambda function.

   The function's code is displayed in the Lambda console..

To explore the API in API Gateway

1. With your project open in the AWS CodeStar console, on the side navigation bar, choose **Project**.
2. In **Project Resources**, in the **ARN** column, choose the link for the Amazon API Gateway API.

   Resources for the API are displayed in the API Gateway console.

Step 3: Test the Web Service

In this step, you test the web service that AWS CodeStar just built and deployed.

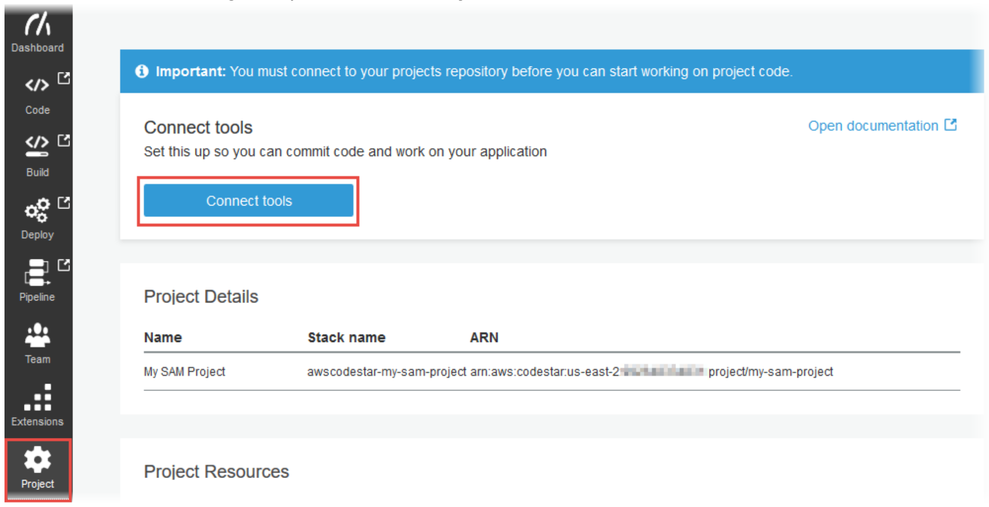
1. With your project still open from the previous step, on the side navigation bar, choose **Dashboard**.
2. On the **Continuous deployment** tile, make sure **Succeeded** is displayed for the **Source**, **Build**, and **Deploy** stages before you continue. This might take several minutes.
3. Choose the link on the **Application endpoints** tile. It should look like **https://*`API_ID`*.execute-api.*`REGION_ID`*.amazonaws.com/Prod/**, where:
   - *`API_ID`* is the ID that API Gateway assigned to the API.
   - *`REGION_ID`* is the ID of the AWS Region.
   - **Prod** is the name of the API deployment stage in API Gateway.

On the new tab that opens in your web browser, the web service displays the following response output:`{"output": "Hello World", "timestamp": "2017-08-30T15:53:42.682839"}`

Step 4: Set Up Your Local Workstation to Edit Project Code

In this step, you set up your local workstation to edit the source code in the AWS CodeStar project. Your local workstation can be a physical or virtual computer running macOS, Windows, or Linux.

1. With your project still open from the previous step, do one of the following:
   - If **You must connect to your project's repository before you can start working on the code** is displayed, choose **Connect Tools**.
   - In the side navigation pane, choose **Project**, and then choose **Connect tools**.



Important: You must connect to your projects repository before you can start working on project code.

Connect tools

Set this up so you can commit code and work on your application

Project Details

Name	Stack name	ARN
My SAM Project	awscodestar-my-sam-project	arn:aws:codestar:us-east-2:[redacted] project/my-sam-project

Project Resources

2. Choose the **Command line tools** tile.

If you have Visual Studio or Eclipse installed, choose the **Visual Studio** or **Eclipse** tile instead, follow the instructions, and then skip to St ep 5: Add Logic to the Web Service.

3. On **Connect to your tools**, for **Operating System**, choose the operating system running on your local workstation.
4. For **Connection Method**, choose **HTTPS**.

   We recommend that you choose HTTPS instead of SSH because HTTPS has fewer setup tasks. If you must use SSH, choose **SSH**, follow the instructions, and then skip to Step 5: Add Logic to the Web Service.
5. Follow the instructions to complete the following tasks:
   a. Set up Git on your local workstation.
   b. Use the IAM console to generate Git credentials for your IAM user.
   c. Clone the project's CodeCommit repository onto your local workstation.

Step 5: Add Logic to the Web Service

In this step, you use your local workstation to add logic to the web service. Specifically, you add a Lambda function and then connect it to the API in API Gateway.

1. On your local workstation, go to the directory that contains the cloned source code repository.
2. In that directory, create a file named`hello.py`. Add the following code, and then save the file:

```
import json def handler(event, context):  data = {    'output':
'Hello ' + event["pathParameters"]["name"]  }  return
{'statusCode': 200,    'body': json.dumps(data),    'headers':
{'Content-Type': 'application/json'}}
```

3. The preceding code outputs the string `Hello` and string the caller sends to the function.
4. In the same directory, open the `template.yml` file. Add the following code to the end of the file, and then save the file:

```
5. Hello:
      Type: AWS::Serverless::Function
      Properties:
        Handler: hello.handler
        Runtime: python2.7
        Role:
          Fn::ImportValue:
            !Join ['-', [!Ref 'ProjectId', !Ref 'AWS::Region',
'LambdaTrustRole']]
        Events:
          GetEvent:
            Type: Api
            Properties:
              Path: /hello/{name}
              Method: get
```

6. Run **git add .** to add your file changes to the staging area of the cloned repository Do not forget the period (**.**), which adds all changed files.
   Note

   If you are using Visual Studio or Eclipse instead of the command line, the instructions for using Git might be different. See the Visual Studio or Eclipse documentation.
7. Run **git commit -m "Added hello.py and updated template.yaml."** to commit your staged files in the cloned repository
8. Run **git push** to push your commit to the remote repository.

After AWS CodeStar detects the push, it instructs CodePipeline to use CodeBuild and AWS SAM to rebuild and redeploy the web service.

AWS SAM gives the new function the name **awscodestar-my-sam-project-lambda-Hello-*RANDOM_ID***, where:

- **my-sam-project** is the ID of the project.
- **Hello** is the function ID, as specified in the `template.yaml` file.
- *RANDOM_ID* is a random ID that AWS SAM assigns to the function for uniqueness.

Step 6: Test the Enhanced Web Service

In this step, you test the enhanced web service that AWS CodeStar built and deployed, based on the logic you added in the previous step.

1. With your project still open in the AWS CodeStar console, on the side navigation bar, choose **Dashboard**.
2. On the **Continuous deployment** tile, make sure the pipeline has run again and that **Succeeded** is displayed for the **Source**, **Build**, and **Deploy** stages before you continue. This might take several minutes.
3. Choose the link on the **Application endpoints** tile. It should look like **https://*API_ID*.execute-api.*REGION_ID*.amazonaws.com/Prod/**, where:
    - *API_ID* is the ID that API Gateway assigned to the API.
    - *REGION_ID* is the ID of the AWS Region.
    - **Prod** is the name of the API deployment stage in API Gateway.

   On the new tab that opens in your web browser, the web service displays the following response output:

   ```
   {"output": "Hello World", "timestamp": "2017-08-30T15:53:42.682839"}
   ```

4. In the tab's address box, add the path **`/hello/`** and your first name to the end of the URL (for example, **https://*API_ID*.execute-api.*REGION_ID*.amazonaws.com/Prod/hello/*YOUR_FIRST_NAME***), and then press **Enter**.

If your first name is Mary, the web service displays the following response output:

```
{"output": "Hello Mary"}
```

Step 7: Add a Unit Test to the Web Service

In this step, you use your local workstation to add a test that AWS CodeStar runs on the web service. This test replaces the manual testing you did earlier.

1. On your local workstation, go to the directory that contains the cloned source code repository.
2. In that directory, create a file named`hello_test.py`. Add the following code, and then save the file.

```
from hello import handler

def test_hello_handler():

  event = {
    'pathParameters': {
      'name': 'testname'
    }
  }

  context = {}

  expected = {
    'body': '{"output": "Hello testname"}',
    'headers': {
      'Content-Type': 'application/json'
    },
    'statusCode': 200
  }

  assert handler(event, context) == expected
```

3. In the same directory, open the `buildspec.yml` file. Replace the file's contents with the following code, and then save the file.

```
version: 0.2


phases:

  install:
    commands:
      - pip install pytest

  pre_build:
    commands:
      - pytest

  build:
    commands:
      - aws cloudformation package --template template.yml
--s3-bucket $S3_BUCKET --output-template template-export.yml

artifacts:
  type: zip
  files:
    - template-export.yml
```

   This build specification instructs CodeBuild to install pytest, the Python test framework, into its build environment. CodeBuild uses pytest to run the unit test. The rest of the build specification is the same as before.
4. Use Git to push these changes to the remote repository.

```
git add .

git commit -m "Added hello_test.py and updated buildspec.yml."

git push
```

Step 8: View Unit Test Results

In this step, you see whether the unit test succeeded or failed.

1. With your project still open in the AWS CodeStar console, on the side navigation bar, choose **Dashboard**.
2. On the **Continuous deployment** tile, make sure the pipeline has run again before you continue. This might take several minutes.

   If the unit test was successful, **Succeeded** is displayed for the **Build** stage.
3. To view the unit test result details, on the **Continuous deployment** tile, in the **Build** stage, choose the **CodeBuild** link.
4. In the CodeBuild console, on the **Build Project: my-sam-project** page, in **Build history**, choose the link in the **Build run** column of the table.
5. On the **my-sam-project:*`BUILD_ID`*** page, in **Build logs**, choose the **View entire log** link.
6. In the Amazon CloudWatch Logs console, look in the log output for a test result similar to the following. In the following test result, the test passed:

```
...
============================= test session starts ==============================
platform linux2 -- Python 2.7.12, pytest-3.2.1, py-1.4.34, pluggy-0.4.0
rootdir: /codebuild/output/src123456789/src, inifile:
collected 1 item

hello_test.py .

=========================== 1 passed in 0.01 seconds ===========================
...
```

If the test failed, there should be details in the log output to help you troubleshoot the failure.

Step 9: Clean Up

In this step, you clean up the project to avoid ongoing charges for this project.

If you want to keep using this project, you can skip this step, but your AWS account might continue to be charged.

1. With your project still open in the AWS CodeStar console, on the side navigation bar, choose **Project**.
2. Choose **Delete project**.
3. Enter the name of the project, keep the **Delete associated resources along with AWS CodeStar project** box selected, and then choose **Delete**.
4. **my-sam-project** is the ID of the project you just deleted.
5. Choose **Empty Bucket**. Enter the name of the bucket, and then choose **Confirm**.
6. Choose **Delete Bucket**. Enter the name of the bucket, and then choose **Confirm**.
7. Now you deleted all resources related to code star.