NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Continuous Deployment - Serverless

Cloud Infrastructure Engineering

Nanyang Technological University & Skills Union - 2022/2023

Course Content

- Self Study Check In
- Recap Prev Session about Serverless Application
- Activity

Q1: What command is used to deploy serverless application?Using serverless & SAM

Q2: What command is used to remove serverless application?Using serverless & SAM

Activity

Instructor

- Ask to use AWS use single region for all learner for easier monitoring serverless

What is Serverless Framework?

Continuous Deployment (CD) is a software development practice that allows developers to **automatically deploy** new versions of an application to production **as soon as they are ready**. It is a key part of the DevOps philosophy, which emphasizes collaboration and automation between development and operations teams.

Serverless computing, on the other hand, is a cloud computing model where the cloud provider manages the underlying infrastructure, and the developer only needs to write and deploy code.

Benefits of Continuous Deployment with Serverless Computing

Continuous deployment with serverless computing offers several benefits, including:

1. **Faster Time-to-Market:** With continuous deployment, new code changes can be deployed to production quickly and easily, allowing businesses to innovate and respond to market needs faster.
2. **Increased Agility:** Continuous deployment allows for rapid iteration and experimentation, which can help businesses stay ahead of the competition.
3. **Improved Quality and Reliability:** Automated testing and deployment processes can help ensure that code changes are thoroughly tested before they are deployed to production, resulting in higher quality and more reliable applications.
4. **Lower Operational Overhead:** Serverless computing allows developers to focus on writing and testing code, while the cloud provider handles the scaling and deployment of functions, resulting in lower operational overhead.

Challenges and Risks of Continuous Deployment with Serverless Computing

While there are many benefits to continuous deployment with serverless computing, there are also several challenges and risks to be aware of. These include:

1. Increased Complexity: Serverless computing can be more complex than traditional server-based deployment models, which can make it more difficult to manage.
2. Security and Compliance Risks: As with any cloud-based deployment model, there are security and compliance risks to be aware of when deploying with serverless computing.
3. Vendor Lock-In: Because serverless computing relies on proprietary cloud infrastructure, there is a risk of vendor lock-in, which can limit the flexibility of the deployment model.

Implementing Continuous Deployment with Serverless Computing

To implement continuous deployment with serverless computing, there are several steps that developers need to take:

1. Use a CI/CD tool: Developers need to use a Continuous Integration/Continuous Deployment (CI/CD) tool that can automatically build and deploy code changes.
2. Proper Code Structuring and Testing: Developers need to ensure that their code is properly structured and tested before deploying it to production.
3. Infrastructure Configuration: Developers need to ensure that their infrastructure is correctly configured to handle the deployment of serverless functions.
4. Proper Security and Compliance Measures: Developers need to implement proper security and compliance measures to mitigate the risks associated with deploying with serverless computing.

Activity - Instructor Demo Serverless Application

Part 1

- Create new repository - Person No. 1
- Clone new repository - All of Group Member
- Add sample code from: 6m-cloud-3.6-cloud-native-application-serverless-i - Person No. 2
- Push sample code to Github - Person No. 2
- Pull latest code the local computer - All of Group Member

Activity - Instructor Demo Serverless Application

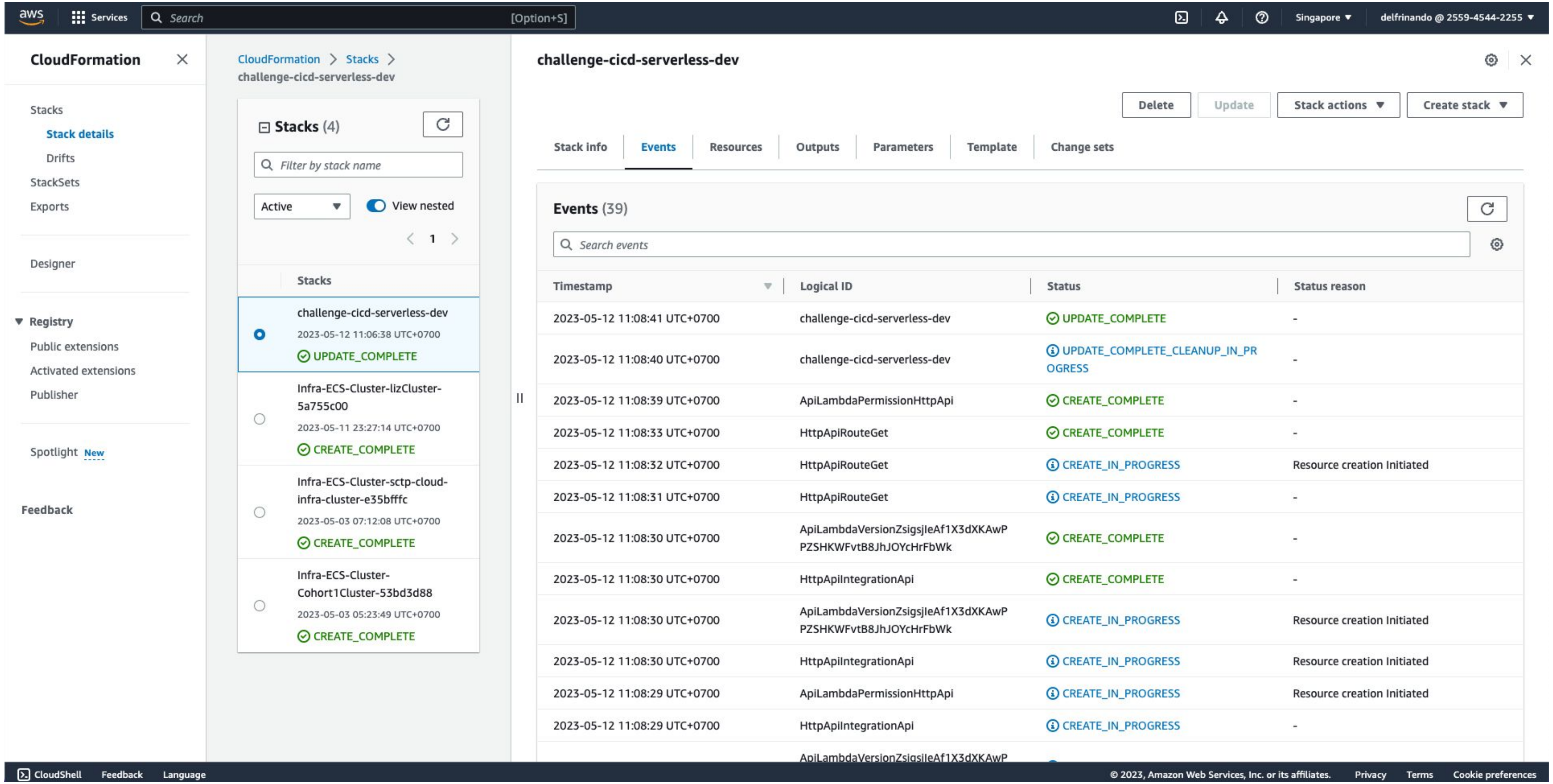
Part 2

- Add new file named “main.yml” workflow on “.github/workflows” -> Person 3
- Update the main.yml with `pre-deploy` -> Person 3
- Push to Repo -> -> Person 3
- Pull -> All Group Member
- Update the main.yml with `install-dependencies` -> Person 1 or Person 4
- Push to Repo -> Person 1 or Person 4
- Pull -> All Group Member

Activity - Instructor Demo Serverless Application

Part 3

- Get Secret -> Person 1
- Add secret -> Person 1
- Add deploy to workflow -> Person 2
- Push to repo -> P2
- Pull -> All



Stacks
challenge-cicd-serverless-dev 2023-05-12 11:06:38 UTC+0700 UPDATE_COMPLETE
Infra-ECS-Cluster-lizCluster-5a755c00 2023-05-11 23:27:14 UTC+0700 CREATE_COMPLETE
Infra-ECS-Cluster-sctp-cloud-infra-cluster-e35bfffc 2023-05-03 07:12:08 UTC+0700 CREATE_COMPLETE
Infra-ECS-Cluster-Cohort1Cluster-53bd3d88 2023-05-03 05:23:49 UTC+0700 CREATE_COMPLETE

challenge-cicd-serverless-dev

Events (39)

Timestamp	Logical ID	Status	Status reason
2023-05-12 11:08:41 UTC+0700	challenge-cicd-serverless-dev	UPDATE_COMPLETE	-
2023-05-12 11:08:40 UTC+0700	challenge-cicd-serverless-dev	UPDATE_COMPLETE_CLEANUP_IN_PROGRESS	-
2023-05-12 11:08:39 UTC+0700	ApiLambdaPermissionHttpApi	CREATE_COMPLETE	-
2023-05-12 11:08:33 UTC+0700	HttpApiRouteGet	CREATE_COMPLETE	-
2023-05-12 11:08:32 UTC+0700	HttpApiRouteGet	CREATE_IN_PROGRESS	Resource creation Initiated
2023-05-12 11:08:31 UTC+0700	HttpApiRouteGet	CREATE_IN_PROGRESS	-
2023-05-12 11:08:30 UTC+0700	ApiLambdaVersionZsigsjIeAf1X3dXKAwPPZSHKWFvtB8JhJOYcHrFbWk	CREATE_COMPLETE	-
2023-05-12 11:08:30 UTC+0700	HttpApiIntegrationApi	CREATE_COMPLETE	-
2023-05-12 11:08:30 UTC+0700	ApiLambdaVersionZsigsjIeAf1X3dXKAwPPZSHKWFvtB8JhJOYcHrFbWk	CREATE_IN_PROGRESS	Resource creation Initiated
2023-05-12 11:08:30 UTC+0700	HttpApiIntegrationApi	CREATE_IN_PROGRESS	Resource creation Initiated
2023-05-12 11:08:29 UTC+0700	ApiLambdaPermissionHttpApi	CREATE_IN_PROGRESS	Resource creation Initiated
2023-05-12 11:08:29 UTC+0700	HttpApiIntegrationApi	CREATE_IN_PROGRESS	-

Activity - Learners

Part 1

- Create new repository
- Add sample code from: 6m-cloud-3.10-continuous-integration
- Run “npm install”
- Test to run the app.

Part 2

- Add workflow on “.github/workflows”
- Add serverless deploy command
- Push Changes
- Check the progress on AWS Github Action & AWS CloudFormation

Challenge

Challenge

Work as Final Project Group.

Spend 45-60 mins for Learners Add Serverless Deployment as part of the Github workflow.

- Create new repository
- Add sample application
- Add unit testing if possible
- Add GitHub Action workflow

Completion criteria

- New repository created
- Each group member added as Collaborators and create feature branch & pull request & merged.
- Sample application is running
- Unit testing is passed
- Able to deploy using GitHub Action workflow

Activity

1. Create new repository, then clone to your local.
2. Open your terminal and change the directory to the cloned repo
3. Run “npm init” on your terminal
   a. New files will be created
   b. Push new files to github using these commands
      i. git add .
      ii. git commit -m “Your commit message”
      iii. git push
4. Install serverless using “npm install serverless”
5. Install serverless offline using “npm install serverless-offline –save-dev”
6. git status => to see the changes/ new files added on our project
7. delete node_modules folder
8. push all changes git add . && git commit -m “Your commit message” && git push
9. add new file “.gitignore” and “`node_modules/`” in it
10. copy code in here: https://github.com/su-ntu-ctp/6m-cloud-3.6-cloud-native-application-serverless-i/blob/main/aws-node-http-api-project/index.js

    to new file named index.js
11. copy code in here: https://github.com/su-ntu-ctp/6m-cloud-3.6-cloud-native-application-serverless-i/blob/main/aws-node-http-api-project/serverless.yml
    to new file named serverless.yml
12. Update line 2 on serverless.yml to change it to your own project name.
13. run locally using: serverless offline start

Activity

1. Create workflow:
    a. Create folder on the root level: “.github/workflows”
    b. Create new file named: main.yml
2. Add workflow code in the main.yml file

```
name: CICD for Serverless Application
run-name: ${{ github.actor }} is doing CICD for serverless application

on:
push:
branches: [ main, "*" ]

jobs:
pre-deploy:
runs-on: ubuntu-latest
steps:
- run: echo "The job is automatically triggered by a ${{ github.event_name }} event."
```

3. Add another jobs on the workflow:

```
install-dependencies:
runs-on: ubuntu-latest
steps:
- name: Check out repository code
uses: actions/checkout@v3
- name: Run installation of dependencies commands
run: npm install
```

Activity

4. Unit testing: Install jest using command “npm install jest –save-dev”
5. Install jest in global: “npm install jest -g”
6. Add new file named: index.test.js and include the following code:
7. Run command: “jest”
8. Change script in package.json to `"test": "jest"`
9. Run “serverless deploy” to run the deployment command
10. Update .gitignore to have “.serverless”
11. Add AWS Key and Secret on github setting

Activity

At the end of the challenge, instructor review the code and show how it should be for adding the deployment in the Github workflow

Activity

Learner:

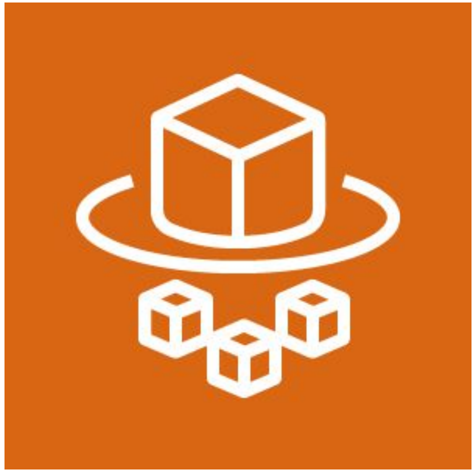
- Clean up AWS.
- Remove/delete/terminate all service/ resources that you created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that you created.
- Check the AWS account after learner clean up.

END

Serverless in AWS



Serverless Framework

What you need to know?

- template specification: **serverless.yml** file
- serverless **cli**
  - create project ⇒ `serverless(or sls)`
  - deploy project ⇒ `sls deploy`

AWS SAM(Serverless Application Model)

What is Serverless Application Model(SAM)?

- **Built on AWS CloudFormation –** Use the AWS CloudFormation syntax directly within your AWS SAM template, taking advantage of its extensive support of resource and property configurations. If you are already familiar with AWS CloudFormation, you don't have to learn a new service to manage your application infrastructure code.
- **An extension of AWS CloudFormation –** AWS SAM offers its own unique syntax that focuses specifically on speeding up serverless development. You can use both the AWS CloudFormation and AWS SAM syntax within the same template.
- **An abstract, shorthand syntax –** Using the AWS SAM syntax, you can define your infrastructure quickly, in fewer lines of code, and with a lower chance of errors. Its syntax is especially curated to abstract away the complexity in defining your serverless application infrastructure.
- **Transformational –** AWS SAM does the complex work of transforming your template into the code necessary to provision your infrastructure through AWS CloudFormation.

Serverless Application Model(SAM)

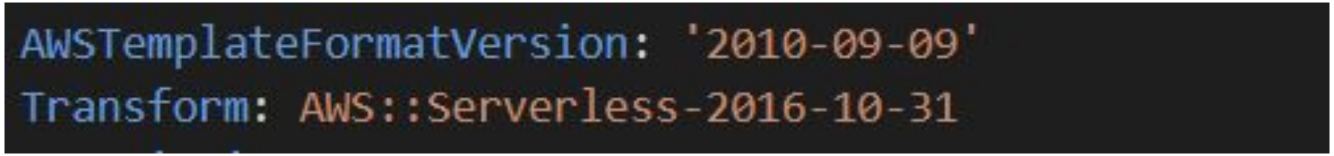
What you need to know?

- Template specification: **eg:- template.yaml** file(YAML or JSON)
- sam **cli**
  - create project ⇒ `sam init`
  - build project ⇒ `sam build`
  - deploy project ⇒ `sam deploy`
- sam **cli config file - samconfig.toml**

Serverless Application Model(SAM) -Template

Transform declaration

- The declaration `Transform: AWS::Serverless-2016-10-31` is required for AWS SAM template files.
- This declaration identifies an AWS CloudFormation template file as an AWS SAM template file.



```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
```

Lambda - Invocation options

1. **Synchronous Invokes**

Synchronous invocations are the most straightforward way to invoke your Lambda functions. In this model, your functions execute immediately when you perform the Lambda Invoke API call.

The Invocation-type flag specifies a value of “**RequestResponse**”. This instructs AWS to execute your Lambda function and wait for the function to complete. When you perform a synchronous invoke, you are responsible for checking the response and determining if there was an error and if you should retry the invoke.

2. **Asynchronous Invokes**

Asynchronous invokes **place your invoke request in Lambda service queue** and we process the requests as they arrive.
You should use AWS X-Ray to review how long your request spent in the service queue by checking the “dwell time” segment.

Synchronous Invokes

Here is a list of services that invoke Lambda functions synchronously:

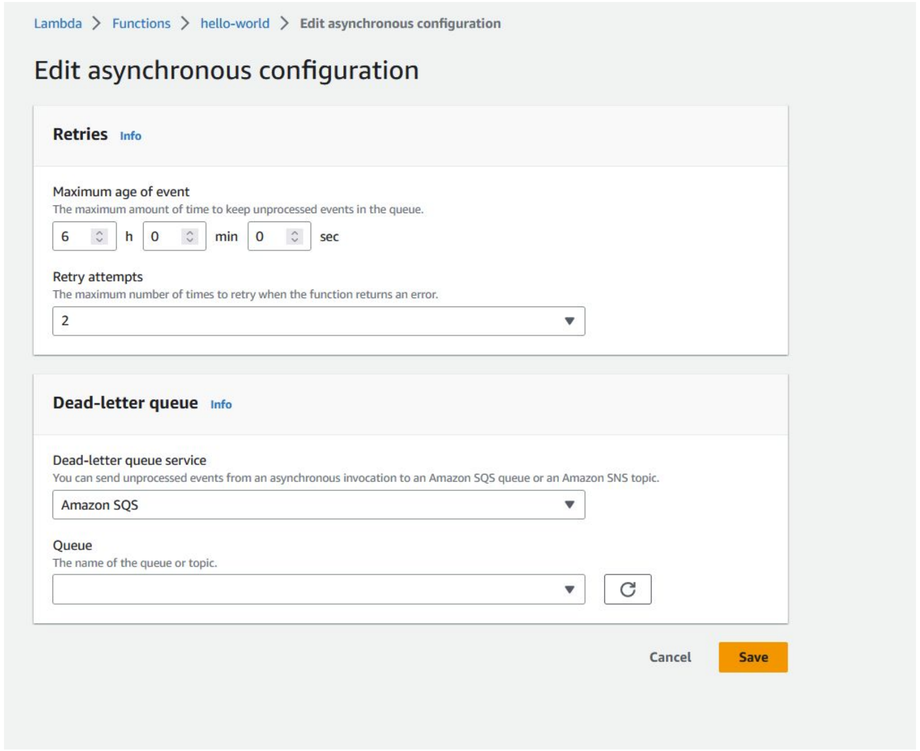
- Amazon API Gateway
- Elastic Load Balancing (Application Load Balancer)
- Amazon Cognito
- Amazon Lex
- Amazon Alexa
- Amazon CloudFront (Lambda@Edge)
- Amazon Kinesis Data Firehose

Asynchronous Invokes

Here is a list of services that invoke Lambda functions asynchronously:

- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon Simple Email Service
- AWS CloudFormation
- Amazon CloudWatch Logs
- Amazon CloudWatch Events
- AWS CodeCommit
- AWS Config

Asynchronous Invokes


Lambda > Functions > hello-world > Edit asynchronous configuration

Edit asynchronous configuration

Retries Info

Maximum age of event
The maximum amount of time to keep unprocessed events in the queue.
6 h 0 min 0 sec

Retry attempts
The maximum number of times to retry when the function returns an error.
2

Dead-letter queue Info

Dead-letter queue service
You can send unprocessed events from an asynchronous invocation to an Amazon SQS queue or an Amazon SNS topic.
Amazon SQS

Queue
The name of the queue or topic.

Cancel Save