IAM Users, Roles & Policies

Add IAM roles & policies needed to meet the assignment objectives to the CloudFormation Template.

Amazon Simple Email Service (SES) Configuration

The following steps are done **manually** and only for the subdomain in the DEMO AWS account.

* [Verify Domain in Amazon SES](http://docs.aws.amazon.com/ses/latest/DeveloperGuide/verify-domains.html)
* [Authenticate Email with DKIM in Amazon SES](http://docs.aws.amazon.com/ses/latest/DeveloperGuide/dkim.html)
* [Move Out of the Amazon SES Sandbox by Requesting Production Access](http://docs.aws.amazon.com/ses/latest/DeveloperGuide/request-production-access.html)

**Warning:**

* **When requesting production access for email, you must submit a genuine case or AWS will deny your request.**
* **AWS support can take 3-5 days to respond to your request so you only have one attempt at getting your request approved to meet the assignment deadline.**
* ***There is nothing I or the TAs can do in getting your AWS request approved.***
* **I do not provide a template or ticket content. Writing up the use case and your needs is one of the objectives of this assignment.**

 Setup GitHub Repository for Lambda Functions

1. [Create](https://help.github.com/articles/create-a-repo/) a new private GitHub repository for serverless function in your GitHub organization.
2. GitHub repository name must be serverless.
3. Grant TAs access to your GitHub repository.
4. Fork the GitHub repository in your namespace. You will do all development work on your fork.
5. All code for serverless should be in this repository.
6. Add appropriate .gitignore to your repository. A collection of useful .gitignore templates can be found [here](https://github.com/github/gitignore).

Amazon Simple Notification Service Configuration

Add Amazon Simple Notification Service (Amazon SNS) topic creation to your CloudFormation configuration.

Email Service & AWS Lambda

Authentication and Access Control for AWS Lambda

* Add [Authentication and Access Control for AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/lambda-auth-and-access-control.html) to your CloudFormation templates.

Web Application Updates

For this assignment, we will update our web application to verify the user's email address on sign-up. The email verification flow should be as follows:

1. The user creates a new account.
2. The web application creates a one-time use token (stored in the DynamoDB, schema/model not included in swagger) and posts a message to the SNS topic. The message includes the following at minimum:
   1. User's email address.
   2. One-time use token.
   3. Message Type
3. The one-time-use token should be valid for only 5 minutes after creation. Use the [DynamoDB TTL feature](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/TTL.html) to implement this.
4. SNS will trigger the Lambda function which will send an email to the user. The email message must include a valid link to your domain (which is environment-specific i.e. in dev, the link should be for the dev subdomain). Example link: <http://prod.domain.tld/v1/verifyUserEmail?email=user@example.com&token=sometoken>
5. The email message should have proper DKIM signature and SPF records.
6. The email message should not end up in the user's SPAM/Junk folder.
7. When the user clicks on the link (while it is valid), the user should be marked as verified.
8. Unverified users should not be able to make any authenticated API call until they have verified their account.
9. We will not implement an API for users to request a new one-time-use token to verify the user account.

Implement Lambda Function

1. Lambda function will be invoked by the SNS notification. Lambda function is responsible for sending emails to the user.
2. As a user, I should receive an email to verify my email address.
3. As a user, I should NOT receive duplicate email messages.
4. Track emails being sent in DynamoDB so that user does not get duplicate emails even if duplicate messages are posted to the SNS topic.

Implement CI/CD Pipeline for Lambda Function

Using the same principles as in previous assignments, implement continuous deployment for Lambda functions. Every commit to the serverless repository should trigger the build and deployment of your updates function to AWS Lambda.

Documentation

Amazon Simple Email Service

* [Getting Started Receiving Email with Amazon SES](https://docs.aws.amazon.com/ses/latest/DeveloperGuide/receiving-email-getting-started.html)
* [Set Up an Email-Receiving Pipeline](https://aws.amazon.com/getting-started/projects/setup-email-receiving-pipeline/)

Amazon Simple Notification & AWS Lambda

* [Using AWS Lambda with Amazon SNS](https://docs.aws.amazon.com/lambda/latest/dg/with-sns-example.html)
* [Invoking Lambda functions using Amazon SNS notifications](https://docs.aws.amazon.com/sns/latest/dg/sns-lambda.html)
* [Creating a .zip Deployment Package (Java)](https://docs.aws.amazon.com/lambda/latest/dg/create-deployment-pkg-zip-java.html)
* [Libraries, samples and tools to help Go developers develop AWS Lambda functions](https://github.com/aws/aws-lambda-go)
* [Lambda Function Examples (receiving-email-action-lambda-example-functions)](https://docs.aws.amazon.com/ses/latest/DeveloperGuide/receiving-email-action-lambda-example-functions.html)
* [Tutorial: Using AWS Lambda with Amazon Simple Notification Service](https://docs.aws.amazon.com/lambda/latest/dg/with-sns-example.html)

AWS CodeDeploy & AWS Lambda

* <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-steps.html#deployment-process-workflow-lambda>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html#appspec-hooks-lambda>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-create-console-lambda.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-example.html#appspec-file-example-lambda>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-resources.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/troubleshooting-deployments-lambda.html>

Submission

The assignment will be considered late if commits are made to the **main** branch after the due date.

1. All work for this assignment must be done on the feature branch in your fork and merged to main when you are dev complete.
2. The feature and main branches must be in sync.
3. Submit your code from **all** repositories in this assignment. **Read the instructions carefully to create your zip file.**
   1. Create a folder with the naming convention **firstname\_lastname\_neuid**
   2. In the **firstname\_lastname\_neuid**clone all of your GitHub (organization) repositories with the **git clone**command. It is important that you clone the repos so that your commit history and branches are preserved.
   3. Once you have cloned all of your repositories, you will create a zip of the **firstname\_lastname\_neuid\_a\_##** directory. The zip file should be **firstname\_lastname\_neuid\_a\_##.zip** where **##** is the assignment number.
   4. Now unzip the zip file in some other directory and confirm the content of the zip files.
   5. Upload the Zip to this assignment.
4. You are allowed to resubmit. If you think there may be an issue with the ZIP file, feel free to submit it again. Only the latest will be used grading.

Grading Guidelines

The following guidelines are for information only. They are subject to change at the discretion of the instructor and TA. 10% penalty will be imposed if the application crashes due to unhandled exceptions/errors.

Previous Assignment Objectives

* TAs must verify that students have resolved all open issues from the previous assignment(s).

Git Repository Content Check (10% Penalty)

* Check the repository for any IDE-specific files. IDE configuration files must not be in the repository.
  + Verify their .gitignore configuration.
* Check the repository for build artifacts such as .class, .jar, .war files and build, node\_modules directory. None of these should be checked into the repository.
* Check for dependencies. Dependencies from the Maven repository or npm should not be committed to the git repository.

Pre-requisite for Demo (5% Penalty)

* Delete all VPCs you may have created in your demo AWS account.

Application Crash (10% Penalty)

* The application should not crash or return *500 Internal Server Errors* during the demo.

Security Check (15% Penalty)

* Check for hard-coded AWS credentials.
* Check IAM policies that grant access to all (\*) resources.
* Check for IAM roles and policies that have AdministratorAccess attached to them.
* Image deletion must be hard deleted in the S3 bucket.

Amazon SES Setup (15%)

1. Verify student’s domain has production access for SES in prod account.
   1. Check Verified Emails for SES and make sure all entries are removed.
2. Verify that DKIM is configured for the domain.
3. Emails should not go to the spam folder.

Infrastructure as Code (20%)

1. Using CloudFormation, create all AWS resources need to run your application. You must use the AMI you have built to launch the EC2 instance.
2. TAs must verify that no resources exist in the default VPC.
3. All IAM roles and policies must be created using CloudFormation. Students cannot make any changes to created resources using the AWS console.
   1. TAs do not have to look at specific roles or policies. The check will be performed indirectly by CI/CD pipeline results.
4. During the demo, note any feature that does not work as it will be an indication of IaC not being implemented correctly.

CI/CD for Lambda (15%)

1. Have student trigger build for web application and AWS Lambda function. The build must run from the main branch. This will ensure that code on the EC2 instance and in the Lambda function is the latest.
   1. Make sure student starts the build from the GitHub console and not by making commits to their GitHub repository.

Web Application, SNS, SES, and Lambda Demo (50%)

1. The user receives an email verification link when a new account is created.
2. Users cannot make any authenticated API calls until the account is verified. A proper HTTP response code must be returned by the application. Try to update a user profile and it should fail.
3. One-time-use tokens expire after 5 minutes.
   1. Configure tokens to expire in 2 mins for demo.
4. Verify email messages do not go to the spam/junk folder.
5. Clicking on the link in the email must mark the user as verified and allow users to make other API calls. Try to set a user profile and it should succeed.
6. Check the link to the web app is for the load balancer and use the correct subdomain for the environment