

## USER MANUAL FOR ALERTING WORKFLOW



**Prepared by:**

**Team-02**

Amir Maharjan (104088013)

Byambadorj Burentogtokh (103133909)

Santosh Pokhrel (104053011)

**August 2023**

## Table of Contents

<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Introduction	1
1.2 Functional requirements	1
<b>Chapter 2 System Requirements</b>	<b>3</b>
2.1 Software Requirements	3
2.2 Hardware Requirements	3
<b>Chapter 3 Configuration</b>	<b>4</b>
3.1 Terraform variables	4
<b>3.1.1</b> Variables to be configured	5
<b>3.1.2</b> Steps to Configure	5
3.2 Review Resources	5
3.3 Cleaning Up	6
<b>Chapter 4 Installation</b>	<b>7</b>
4.1 Deployment	7
4.2 Clean Up	7
<b>Chapter 5 Code Structure</b>	<b>8</b>
5.1 Cost Metric Lambda Function	8
5.2 Cloud watch Alarms and SNS Topic	10
5.3 Testing Cost Metric Lambda Function	11
5.4 Monitoring the Cloud watch	11
5.5 SNS Topic	12
5.6 Response Data From SNS Topic	12
5.7 Python Script : notifier.py	14
5.8 Terraform Configuration: notifier.tf	15
5.9 Terraform Variables: variables.tf	18
<b>Chapter 6 Troubleshooting</b>	<b>20</b>

6.1 Issue: Lambda Function Execution Failure .....	20
6.2 Issue: Email or Slack Notifications Not Received.....	20
6.3 Issue: Missing Terraform Variables.....	20
<b>Chapter 7 FAQs (Frequently Asked Questions) .....</b>	<b>22</b>
<b>Chapter 8 Contact Information .....</b>	<b>23</b>
<b>References.....</b>	<b>24</b>

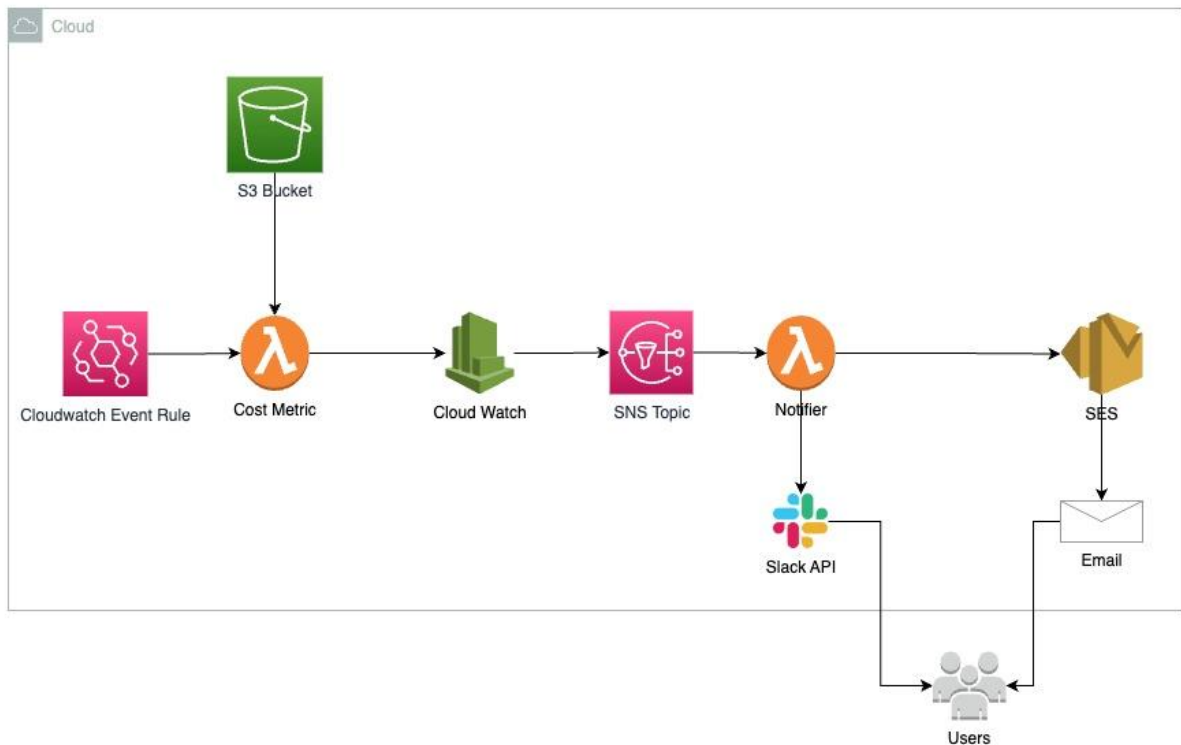
# Chapter 1 Introduction

## 1.1 Introduction

Welcome to the user manual for the Alert Feature of the XC3! This manual is designed to provide developers with all the information they need to understand, install, configure, and effectively use the codebase. The main purpose of this feature is to send an alert to the end user on three levels: AWS account level, AWS resources level and IAM user level when the pre defined cost threshold crossed.

## 1.2 Architecture

This alerting workflow starts with a CloudWatch event rule triggering the Cost Metric Lambda. This Lambda collects cost data in JSON format from an S3 bucket, calculates a cost percentage, and sends it to a CloudWatch metric. CloudWatch watches this metric and activates an alarm cost percentage is greater than the threshold. This alarm sends a message to an SNS topic. The SNS topic then forwards the JSON message to the Notifier Lambda, which processes and formats it into email and Slack notifications for efficient alerting via both channels.



## 1.3 Functional requirements

- Implementing an end-to-end alerting mechanism that covers all stages of the alerting process.

- Defining the specific threshold criteria or conditions that trigger the alerts.
- Generating alerts promptly when the threshold is met.
- Sending the alert information to relevant communication channels, such as email, SMS, or instant messaging services.
- Ensuring the proper integration and functionality of the communication channels to transmit the alerts effectively.
- Providing a means to configure and customize the alerting system to accommodate different requirements.
- Handling and addressing potential errors or exceptions that may occur during the alerting process.
- Ensuring the scalability and performance of the alerting mechanism to handle large volumes of data and alerts.

Inputs: Make a parameterized variable for the threshold

Outputs: Alerts

## **Chapter 2** System Requirements

### **2.1** Software Requirements

The bare minimum software requirements to run the AlertFeature are:

- Terraform 1.0+
- Python 3.9
- AWScli
- AWS cloud access with administrative privilege
- IDE/Code editor

### **2.2** Hardware Requirements

The bare minimum hardware requirements to run the AlertFeature are:

- Personal computer with at least 4GB RAM
- Internet Connection

## Chapter 3 Configuration

This chapter emphasizes on how to configure the AWS Lambda function and associated resources using Terraform. The provided configuration allows the developers to set up a notification mechanism for CloudWatch Alarms through email and Slack notifications.

This chapter emphasizes on how to configure the AWS Lambda function and associated resources using Terraform. The provided configuration allows the developers to set up a notification mechanism for CloudWatch Alarms through email and Slack notifications.

### 3.1 Terraform variables

1. Navigate to xc3/infrastructure and configure the terraform.auto.tf.vars file.

```
maximum_budget      = 5.0
threshold            = 10
cloudwatch_namespace = "CustomCostMetric"
metric_name          = "CostPercentageMetric"
```

- Update the variables according to the requirements of your project.
- Maximum Budget should be the maximum budget for your project.
- Threshold should be the desired threshold to check the cost and trigger the alarm
- Cloud watch Namespace is the space to hold all the cloud watch metrics
- Metric Name is the value for the total cost metric name

2. Once variables are configured, execute the following commands:

```
terraform plan -var-file=terraform.auto.tfvars
terraform apply -var-file=terraform.auto.tfvars
```

Enter yes after being prompted to enter.

The successful apply should see the infrastructures for alerting feature added to the existing XC3 infrastructure. The configuration uses Terraform variables to customize the behavior of

the resources. We will have to Review and adjust these variables according to the requirements:

File name: variables.tf

### 3.1.1 Variables to be configured

1. sender\_email: The sender's email address for notifications. Default: swinabs@gmail.com
2. recipient\_email: The recipient's email address for notifications. Default: swinabs@gmail.com
3. region: The AWS region where the resources will be deployed. Default: ap-southeast-2
4. slack\_channel\_url: The URL of the Slack channel for notifications. Default: *(leave empty)*

### 3.1.2 Steps to Configure

Follow these steps to configure the notification mechanism using AWS Lambda and Terraform:

1. Edit Variables: Open the variables.tf file and adjust the values of the variables based on your preferences. These variables will customize the notifications and resource settings.
2. Python Notification Script (notifier.py): The provided Python script defines the logic for handling CloudWatch Alarm notifications. You can customize this script to format notifications, change the content, or add additional actions.
3. Deploy the Configuration: Open your terminal and navigate to the directory containing the configuration files. Run the following Terraform commands:

```
terraform init
terraform plan
terraform apply
```

Terraform will initialize, create an execution plan, and apply the plan to create the AWS Lambda function, IAM roles, policies, and other required resources.

## 3.2 Review Resources

Once Terraform applies the configuration, review the created resources in your AWS Management Console to ensure they match your expectations.



### 3.3 Cleaning Up

When you no longer need the configured resources, use the following command to destroy them and clean up:

```
terraform destroy
```

## Chapter 4 Installation

This chapter guides you through the installation process of the required tools and resources to set up CloudWatch Alarm notifications using AWS Lambda and Terraform.

### 4.1 Deployment

With the code and configuration customized, follow these steps to deploy the CloudWatch Alarm notification system:

1. Open a terminal window and navigate to the directory containing the downloaded code and configuration files.
2. Run the following commands in sequence to deploy the resources:

```
terraform init
terraform plan
terraform apply
```

2. These commands initialize Terraform, create an execution plan, and apply the plan to provision AWS Lambda, IAM roles, policies, and other resources.
3. Review the resources created in your AWS Management Console to ensure they match your expectations.

### 4.2 Clean Up

When you're finished with the CloudWatch Alarm notification system, follow these steps to clean up:

1. In the terminal, navigate to the same directory as before and run the following command:

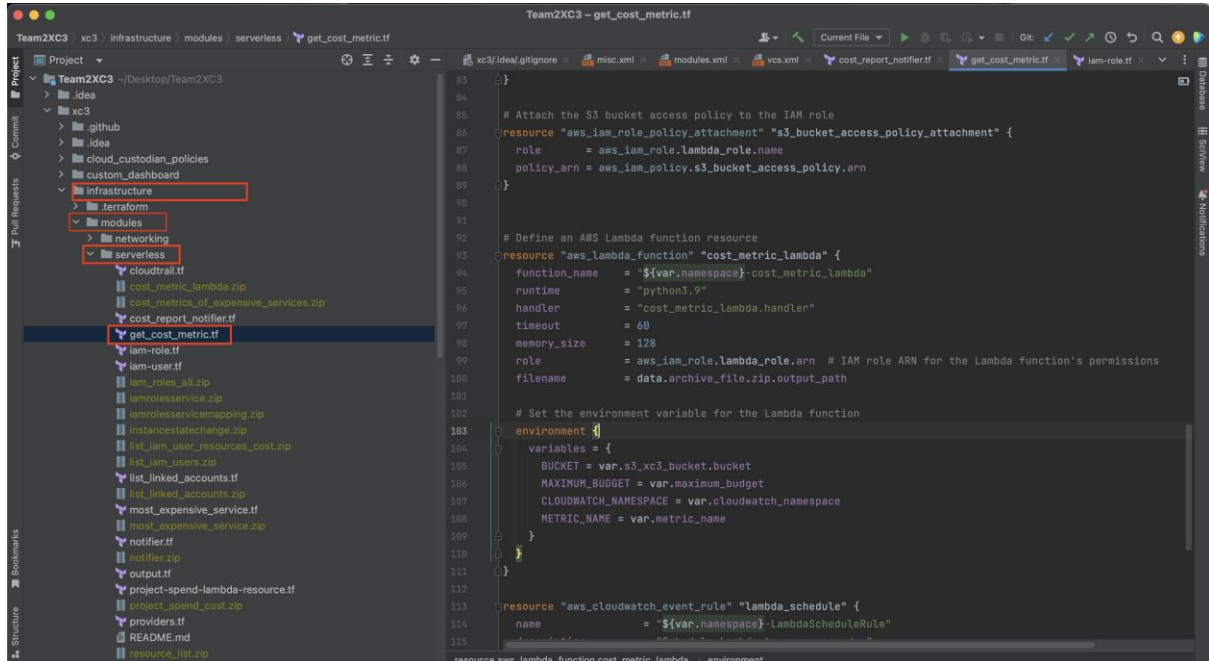
```
terraform destroy
```

1. This command will remove all the resources created by Terraform.

## Chapter 5 Code Structure

### 5.1 Cost Metric Lambda Function

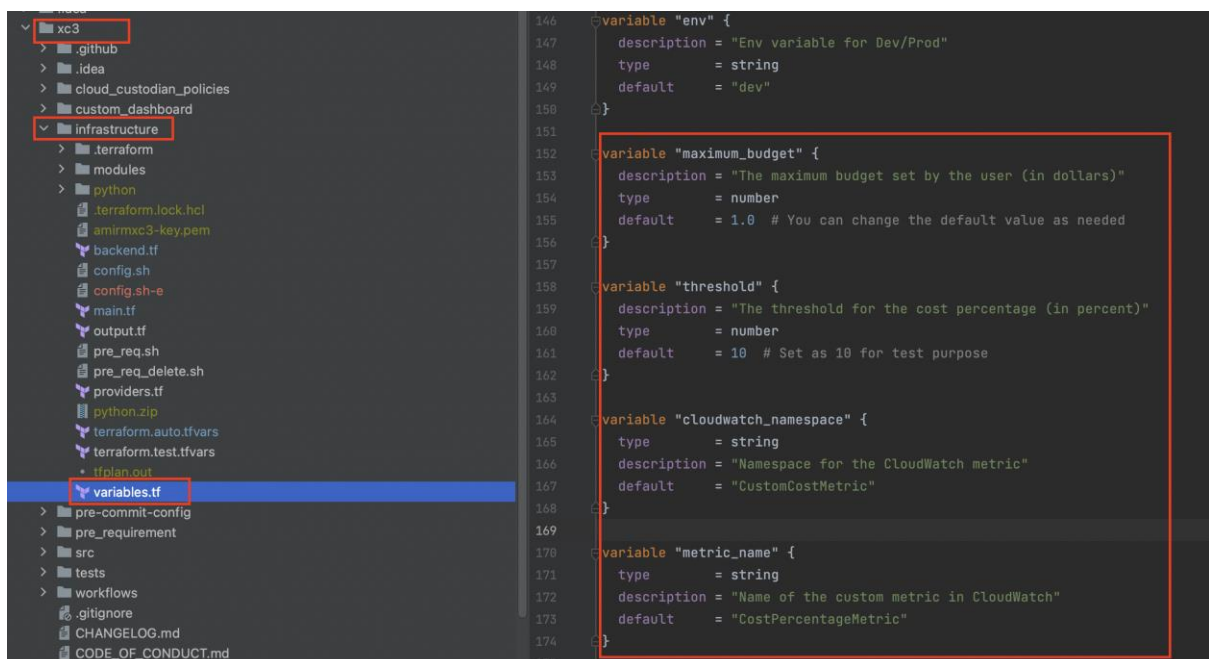
Terraform file with resources for creating lambda function and required IAM roles and policies, cloud watch event is inside the **infrastructure->modules->serverless** folder with file name **get\_cost\_metric.tf**



```
83 }
84
85 # Attach the S3 bucket access policy to the IAM role
86 resource "aws_iam_role_policy_attachment" "s3_bucket_access_policy_attachment" {
87   role       = aws_iam_role.lambda_role.name
88   policy_arn = aws_iam_policy.s3_bucket_access_policy.arn
89 }
90
91 # Define an AWS Lambda function resource
92 resource "aws_lambda_function" "cost_metric_lambda" {
93   function_name = "${var.namespace}.cost_metric_lambda"
94   runtime       = "python3.9"
95   handler       = "cost_metric_lambda.handler"
96   timeout       = 60
97   memory_size   = 128
98   role          = aws_iam_role.lambda_role.arn # IAM role ARN for the Lambda function's permissions
99   filename      = data.archive_file.zip.output_path
100
101   # Set the environment variable for the Lambda function
102   environment {
103     variables = {
104       BUCKET = var.s3_xc3_bucket.bucket
105       MAXIMUM_BUDGET = var.maximum_budget
106       CLOUDWATCH_NAMESPACE = var.cloudwatch_namespace
107       METRIC_NAME = var.metric_name
108     }
109   }
110 }
111
112 resource "aws_cloudwatch_event_rule" "lambda_schedule" {
113   name = "${var.namespace}-lambdaScheduleRule"
114 }
115
```

The variables that are used in this file are firstly listed in the variables.tf file along with all other necessary variables for the alerting feature.

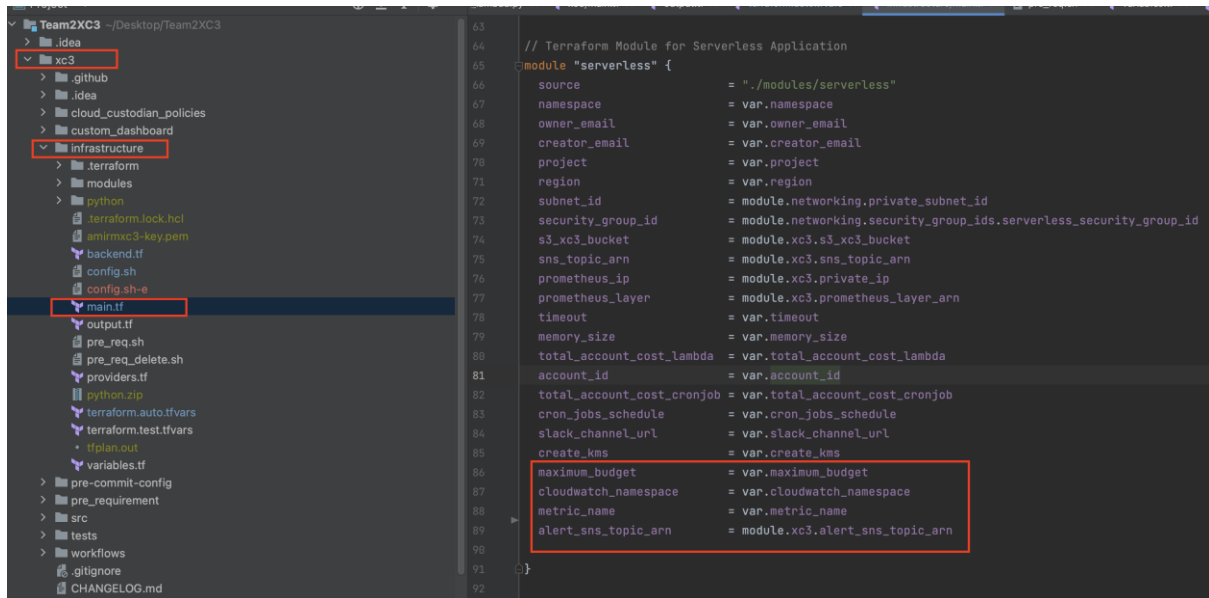
**xc3->infrastructure->variables.tf**



```
146 variable "env" {
147   description = "Env variable for Dev/Prod"
148   type        = string
149   default     = "dev"
150 }
151
152 variable "maximum_budget" {
153   description = "The maximum budget set by the user (in dollars)"
154   type        = number
155   default     = 1.0 # You can change the default value as needed
156 }
157
158 variable "threshold" {
159   description = "The threshold for the cost percentage (in percent)"
160   type        = number
161   default     = 10 # Set as 10 for test purpose
162 }
163
164 variable "cloudwatch_namespace" {
165   type        = string
166   description = "Namespace for the CloudWatch metric"
167   default     = "CustomCostMetric"
168 }
169
170 variable "metric_name" {
171   type        = string
172   description = "Name of the custom metric in CloudWatch"
173   default     = "CostPercentageMetric"
174 }
175
```

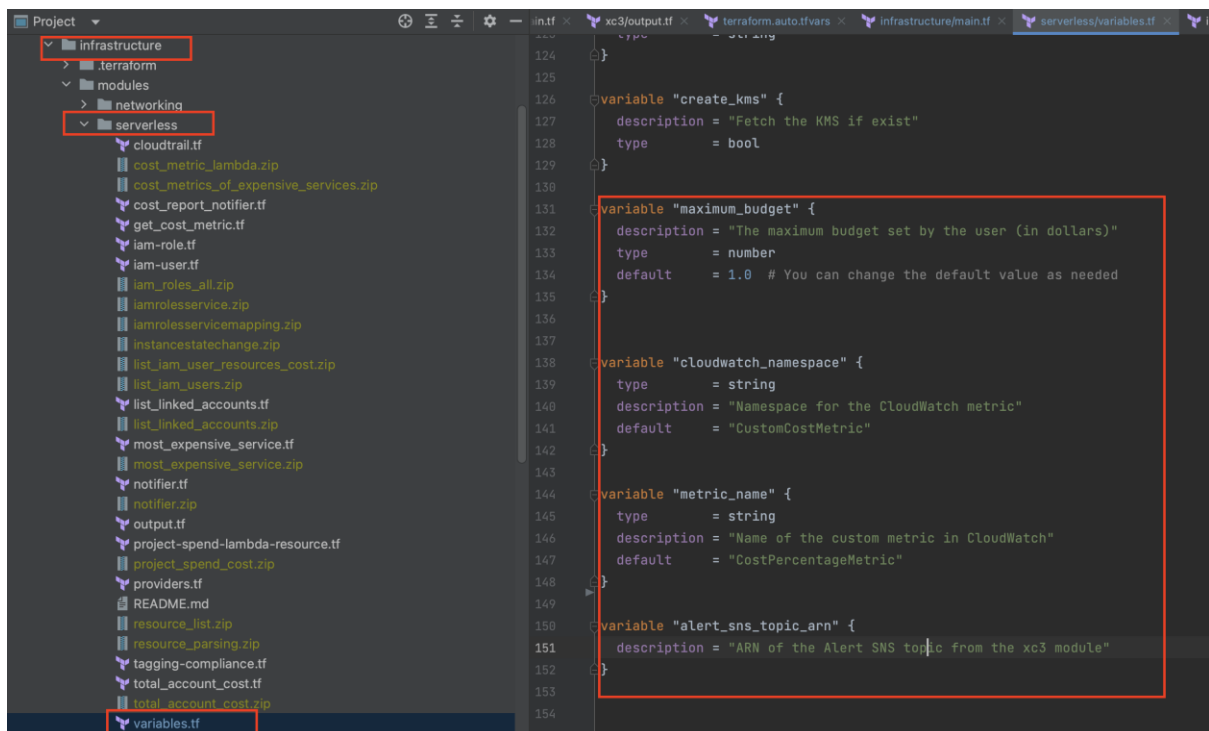
Then only the variables needed for the module should be listed and values should be passed under the serverless module in the file.

#### xc3->infrastructure->main.tf



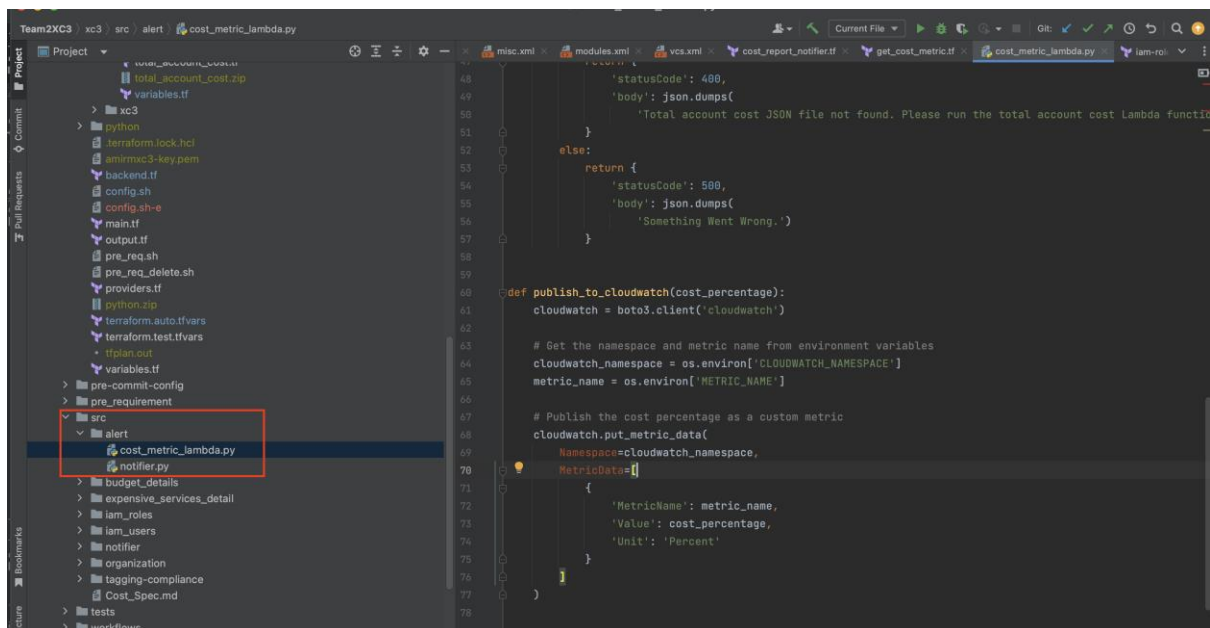
After that, again we need to list the variables that will be used in the serverless module and have received the values from the main.tf file. It should be done in the file

#### xc3->infrastructure->serverless->variables.tf



The python file that is necessary to create lambda function is placed inside

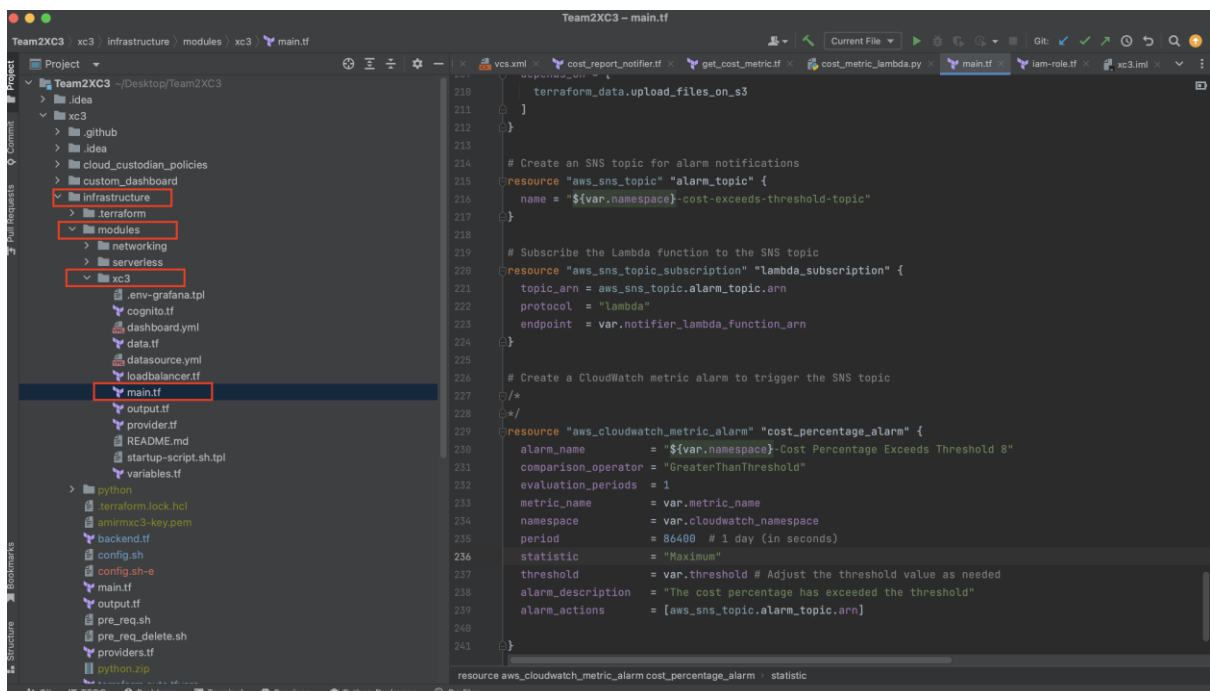
src>alert folder with name **cost\_metric\_lambda.py**



## 5.2 Cloud watch Alarms and SNS Topic

Terraform file for creating resources SNS topic and Cloud watch Alarms are placed under module xc3

infrastructure->modules->xc3 folder with name **main.tf**

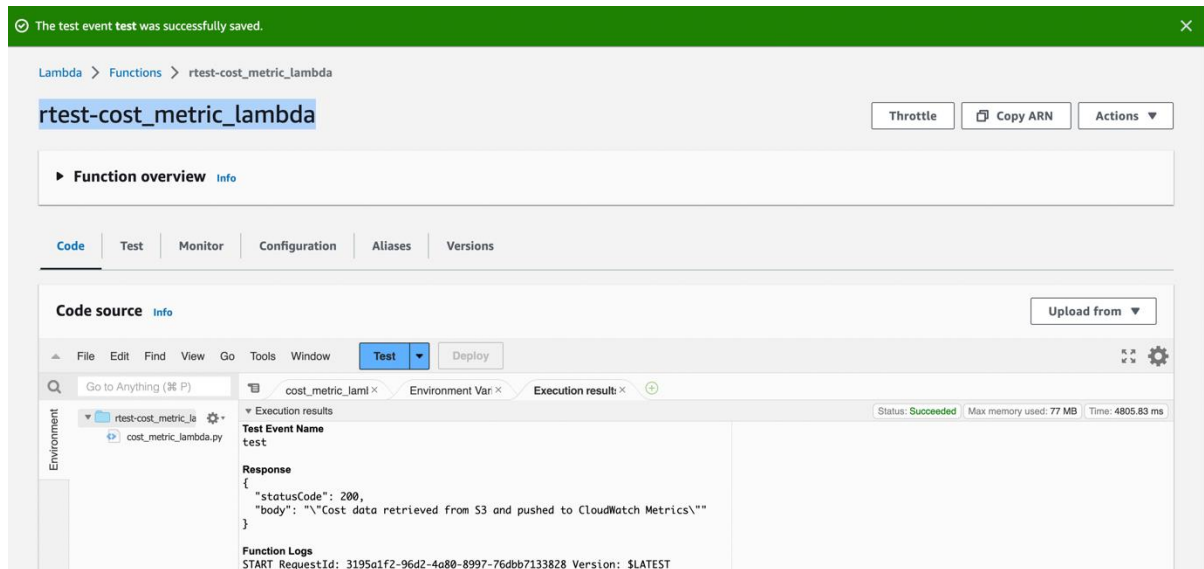


## 5.3 Testing Cost Metric Lambda Function

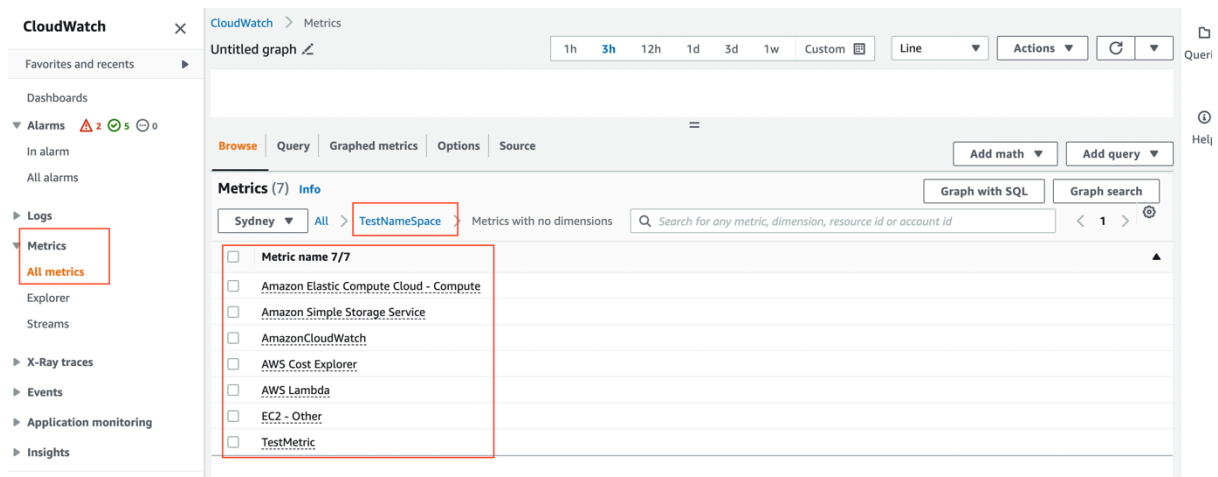
If you want to test the lambda function to calculate the cost percentage and push to cloud watch metrics, you can follow the following steps

**Function name: {namespace}-cost\_metric\_lambda**

Select the "Test" option and set up a test using the default test JSON. Provide a name for your test and run it.



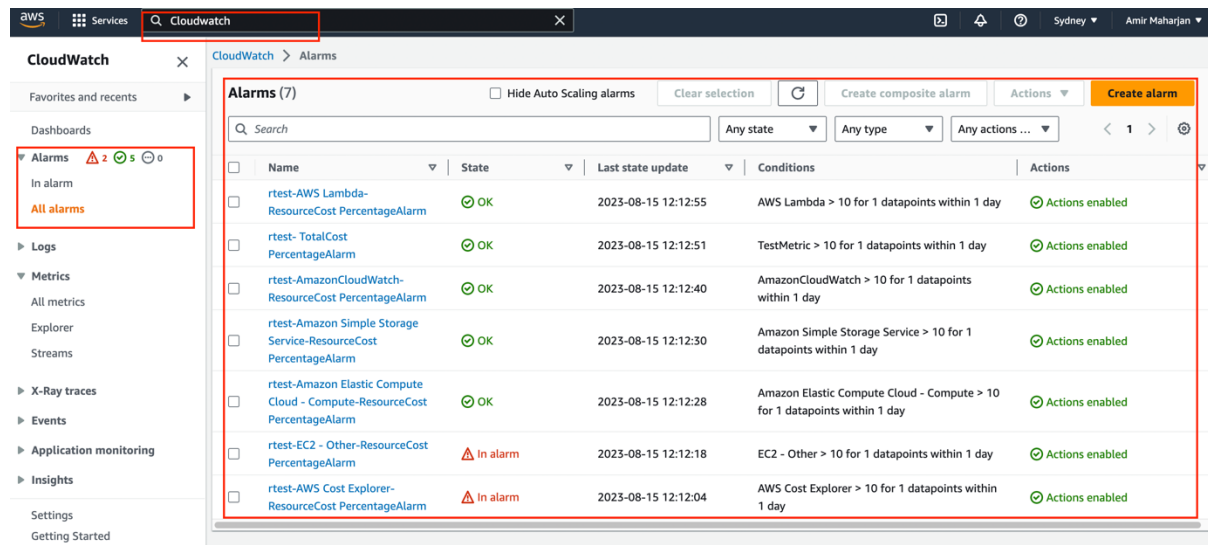
After the successful run of cost metric lambda you will be able to see the metrics at the Cloud watch metrics under the Cloud watch namespace that you have provided in **terraform.auto.tf.vars**



## 5.4 Monitoring the Cloud watch

You can search Cloud watch in top search bar to get the Cloud watch Alarms, then click All Alarms to view all existing alarms.

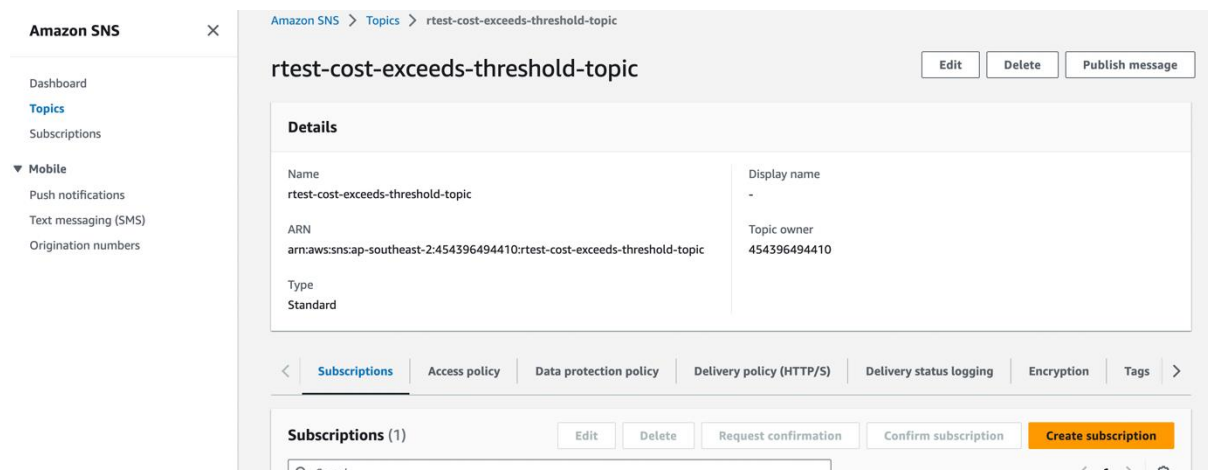
The alarms with state with Ok mean the cost data related to the specific alarms hasn't crossed the threshold and the alarms with In Alarm means the alarms has been triggered as the cost data is greater than the threshold mentioned.



## 5.5 SNS Topic

After alarm is triggered by cloud watch the message is sent from the cloud watch to SNS topic with the details of the Alarm Triggered. The SNS topic can be viewed at

**SNS Topic Name: {namespace}-cost-exceeds-threshold-topic**



## 5.6 Response Data From SNS Topic

After message is received from Cloud watch SNS Topic sends the message to the lambda function that is subscribed to it. The message format in which the SNS topic sends can be seen below.





## 5.7 Python Script : notifier.py

```
# Import necessary libraries
import boto3
import json
import os
import urllib3

def lambda_handler(event, context):
    # Extract the SNS message from the Lambda event
    sns_message = json.loads(event['Records'][0]['Sns']['Message'])

    # Extract relevant details from the SNS message
    # ... (Details extraction code)

    # Compose the email subject and body
    # ... (HTML email body composition)

    # Send the email
    send_email(subject, body)
    send_slack(alarm_name, alarm_description, aws_account_id, region, threshold, iam_user)

    return "Email and Slack notifications sent successfully"

def send_email(subject, body):
    # Configure the email sender
    # ... (Email sender configuration)

    # Create an AWS Simple Email Service (SES) client
    # ... (SES client creation)

    # Send the email
    response = ses.send_email(
        Source=sender_email,
        Destination={'ToAddresses': [recipient_email]},
        Message={
            'Subject': {'Data': subject},
            'Body': {'Html': {'Data': body}}
        }
    )

    return response

def send_slack(alarm_name, alarm_description, aws_account_id, region, threshold, iam_user):
    # Configure HTTP Pool Manager
    # ... (HTTP configuration)

    # Prepare Slack notification message
    # ... (Slack notification message composition)

    # Send the Slack notification
    try:
        r = http.request(
            "POST",
            slack_url,
            body=json.dumps(messages),
            headers={"Content-Type": "application/json"}
        )
        print("Slack notification sent! Status code:", r.status)
        return True
    except Exception as e:
        print("Failed to send Slack notification:", str(e))
        return False
```

This is a Python script designed to be used as an AWS Lambda function. The Lambda function is meant to handle CloudWatch Alarm notifications by sending email and Slack notifications when a certain threshold is breached. Here's a breakdown of its purpose and functionality:

- **Import Statements:** The script imports necessary libraries/modules for AWS services, JSON handling, operating system interactions, and HTTP requests.

- **Lambda Handler Function (``lambda_handler``):** This function is the entry point for the Lambda execution. It takes two parameters, ``event`` and ``context``, which are provided by the Lambda service when the function is triggered.
- **Extracting SNS Message:** The SNS message is extracted from the ``event`` parameter. SNS (Simple Notification Service) is a messaging service used for sending notifications to various endpoints, including email, SMS, and more.
- **Extracting Alarm Details:** The script extracts relevant details from the SNS message, such as alarm name, description, AWS account ID, region, and threshold.
- **Composing Email Body:** The script composes an HTML-formatted email body that contains the extracted alarm details. The body includes stylized CSS and information presented in a table format.
- **Sending Email:** The ``send_email`` function sends the composed email using AWS SES (Simple Email Service). It includes the subject and HTML body of the email.
- **Sending Slack Notification:** The ``send_slack`` function sends a notification to a Slack channel using an HTTP POST request. It formats the message with the extracted alarm details.
- **Return Statement:** The Lambda handler function returns a success message indicating that the email and Slack notifications have been sent successfully.
- **Send Email and Slack Notifications:** The script calls the ``send_email`` and ``send_slack`` functions to send the notifications.

In summary, this code sets up an AWS Lambda function that is triggered when a CloudWatch Alarm is triggered. The Lambda function sends an email and a Slack notification to notify a user or team about the alarm details, such as the breach of a certain threshold in AWS account costs. The email content is formatted using HTML, and the Slack message is sent via an HTTP POST request.

## 5.8 Terraform Configuration: `notifier.tf`

The `notifier.tf` file defines the infrastructure resources required for the AWS Lambda function, IAM roles, policies, and permissions.

This Terraform code is used to create an AWS Lambda function along with the required IAM roles and permissions to set up a notification mechanism for CloudWatch Alarms using AWS Lambda. Let's break down the purpose of each section of the code:

- Create an Archive File (`data "archive_file" "lambda_zip" { ... }`): This section creates a zip archive file containing the Python script "notifier.py" (presumably containing notification logic) that will be uploaded and used by the Lambda function. The script is compressed into a zip file named "notifier.zip."
- Create an AWS Lambda Function (`resource "aws_lambda_function" "notifier" { ... }`): This section defines an AWS Lambda function named "\${var.namespace}-notifier". It specifies various configuration options for the Lambda function, including runtime, handler, timeout, memory size, role, and environment variables. The Lambda function will use the archive file created in the previous step as its source code.
- Create an IAM Role for the Lambda Function (`resource "aws_iam_role" "notifier_role" { ... }`): This section creates an IAM role named "\${var.namespace}-notifier\_role" that will be assumed by the Lambda function. The trust policy allows the Lambda service to assume this role.
- Attach Policies to the IAM Role (`resource "aws_iam_role_policy_attachment" ...``): These sections attach various policies to the IAM role created in step 3. These policies include the AWS managed policy "AWSLambdaBasicExecutionRole" (providing basic execution permissions for Lambda) and a custom policy that grants permission to send emails via Amazon SES. Additionally, the policy "AmazonSESFullAccess" is attached to allow full access to Amazon Simple Email Service (SES).
- Add SNS Trigger Permission (`resource "aws_lambda_permission" ...``): This section grants permission to the Lambda function to be triggered by an Amazon SNS topic. The Lambda function will be allowed to execute when the SNS topic (likely used for CloudWatch Alarms) publishes messages to it.

Overall, the code's purpose is to set up an AWS Lambda function that can be triggered by CloudWatch Alarms through an SNS topic. The Lambda function handles notifications by sending emails and potentially other actions, and it is appropriately configured with IAM roles, policies, and permissions to execute these actions securely within the AWS environment. The use of Terraform allows for infrastructure-as-code management and provisioning of these resources.

```

data "archive_file" "lambda_zip" {
  type      = "zip"
  source_file = "notifier.py"
  output_path = "notifier.zip"
}

resource "aws_lambda_function" "notifier" {
  function_name = "${var.namespace}-notifier"
  runtime       = "python3.9"
  handler       = "notifier.lambda_handler"
  timeout       = 60
  memory_size   = 128
  role          = aws_iam_role.notifier_role.arn
  filename      = data.archive_file.lambda_zip.output_path

  environment {
    variables = {
      sender_email = var.sender_email
      recipient_email = var.recipient_email
      region = var.region
      slack_channel_url = var.slack_channel_url
    }
  }
}

# Create an IAM role for the Lambda function
resource "aws_iam_role" "notifier_role" {
  name = "${var.namespace}-notifier_role"

  # Define the trust policy to allow the Lambda service to assume this role
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
  EOF
}

# Attach the necessary policies to the IAM role
resource "aws_iam_role_policy_attachment" "lambda_basic_execution_attachment" {
  role       = aws_iam_role.notifier_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
}

# Define a custom IAM policy for Send Email permissions
resource "aws_iam_policy" "notifier_custom_policy" {
  name        = "${var.namespace}-CombinedLambdaCustomPolicy"
  description = "Custom IAM policy for combined Lambda function"

  # Define the policy document allowing the "ses:SendEmail" action
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ses:SendEmail"
      ],
      "Resource": "*"
    }
  ]
}
  EOF
}

# Attach the custom policy to the IAM role
resource "aws_iam_role_policy_attachment" "notifier_custom_policy_attachment" {
  role       = aws_iam_role.notifier_role.name
  policy_arn = aws_iam_policy.notifier_custom_policy.arn
}

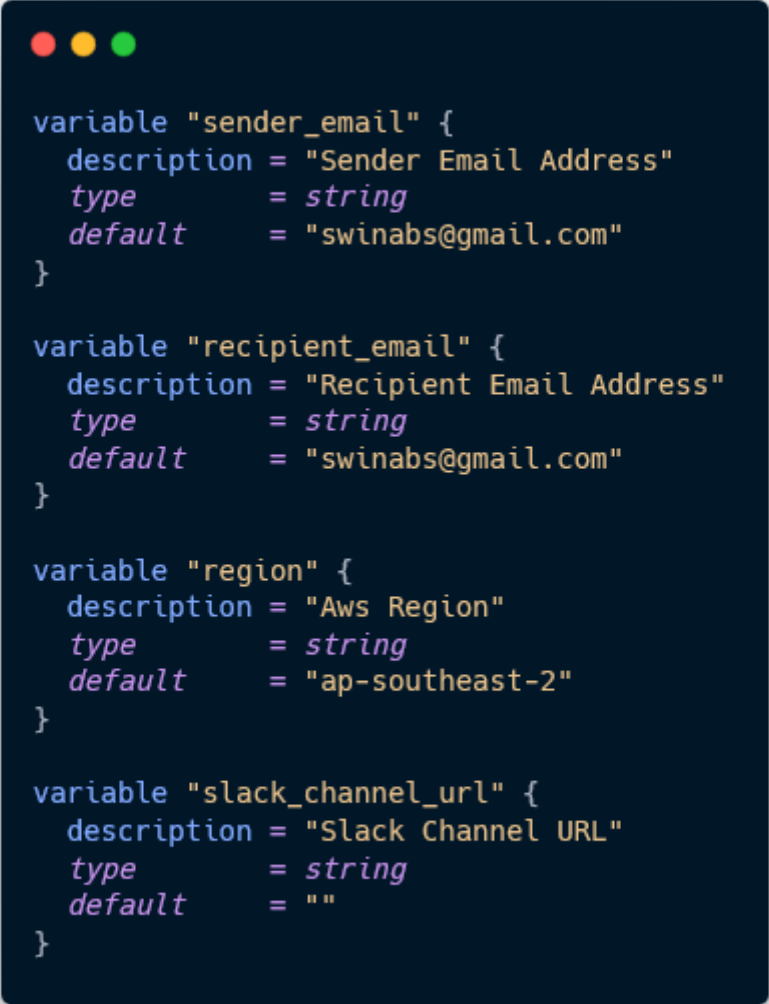
# Attach the Amazon SES policy to the IAM role
resource "aws_iam_role_policy_attachment" "lambda_ses_policy_attachment" {
  role       = aws_iam_role.notifier_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonSESFullAccess"
}

# Add the SNS topic trigger to the Lambda function
resource "aws_lambda_permission" "lambda_sns_trigger_permission" {
  statement_id = "AllowExecutionFromSNS"
  action       = "lambda:InvokeFunction"
  function_name = aws_lambda_function.notifier.function_name
  principal    = "sns.amazonaws.com"
  source_arn    = aws_sns_topic.alarm_topic.arn
}

```

## 5.9 Terraform Variables: variables.tf

The variables.tf file defines input variables that customize the behavior of the Terraform configuration.



```
variable "sender_email" {  
  description = "Sender Email Address"  
  type        = string  
  default     = "swinabs@gmail.com"  
}  
  
variable "recipient_email" {  
  description = "Recipient Email Address"  
  type        = string  
  default     = "swinabs@gmail.com"  
}  
  
variable "region" {  
  description = "Aws Region"  
  type        = string  
  default     = "ap-southeast-2"  
}  
  
variable "slack_channel_url" {  
  description = "Slack Channel URL"  
  type        = string  
  default     = ""  
}
```

This code defines Terraform variables that can be used to customize and parameterize the configuration of infrastructure resources. Variables in Terraform allow you to make your code more flexible and reusable by separating the values that can change from the actual resource definitions.

Here's the purpose of each variable defined in the code:

`sender_email`: This variable is used to specify the email address of the sender. It is intended to be used as a parameter in the Terraform code to provide the sender's email address for notifications or other purposes.

`recipient_email`: This variable is used to specify the email address of the recipient. Similar to the sender's email, it is intended to be used as a parameter in the Terraform code to provide the recipient's email address for notifications or other uses.

`region`: This variable is used to specify the AWS region where the infrastructure resources will be deployed. It is intended to allow users to choose the region in which their resources will be provisioned.

`slack_channel_url`: This variable is used to specify the URL of a Slack channel. It is intended to be used as a parameter to provide the URL of the Slack channel where notifications or messages should be sent.

The ``description`` field for each variable provides a brief explanation of what each variable is used for, aiding in documentation and understanding. The ``type`` field specifies the data type of the variable, such as string, number, or boolean. The ``default`` field provides a default value that will be used if no value is explicitly provided when running Terraform.

By defining these variables, you can write more flexible and reusable Terraform configurations. Users can customize the behavior of the infrastructure by providing different values for these variables without modifying the main resource definitions. This promotes consistency, makes it easier to manage changes, and allows you to reuse the same Terraform code with different configurations.

## Chapter 6 Troubleshooting

If you encounter any issues while setting up or using the CloudWatch Alarm notification mechanism with the provided Terraform configuration and `notifier.py` script, refer to the following troubleshooting tips to help resolve common problems.

### 6.1 Issue: Lambda Function Execution Failure

Solution:

1. Check the CloudWatch Logs for the Lambda function to identify any error messages or exceptions thrown during execution.
2. Review the IAM role permissions associated with the Lambda function. Ensure that it has necessary permissions to execute Lambda functions, access SES, and send messages to the specified Slack channel.
3. Verify that the Python script `notifier.py` is correctly formatted and contains the necessary code. Any syntax errors or missing modules could lead to execution failures.

### 6.2 Issue: Email or Slack Notifications Not Received

Solution:

1. Confirm that the email sender's and recipient's email addresses are correctly provided in the Terraform variables. Make sure they are valid email addresses.
2. Ensure that the AWS Simple Email Service (SES) is correctly configured in your AWS account, and the sender's email address is verified.
3. Double-check the Slack channel URL provided in the variables. Ensure it is the correct URL for the intended Slack channel.

### 6.3 Issue: Missing Terraform Variables

Solution:

1. Ensure that you have correctly defined and set the required Terraform variables in your `variables.tf` file. Review the variable names and descriptions to ensure accuracy.
2. Check that you have provided appropriate default values for the variables, or you have provided custom values when executing Terraform commands.

#### **6.4 Issue:** If you receive the error message: “JSON file not found.

Please run associated Lambda functions first. Then please ensure you have run the associated lambda functions that creates the json file and stores them in S3 bucket. The json file must be under {bucket/cost-metrics folder.

#### **6.5 Issue:** If you see the “Insufficient Data“ in the Cloud watch alarms there could be the several reasons

1. The Cost Metric Lambda function has not run yet as of result there is no metric for the alarms.
2. Please give few mins as cloud watch alarm takes a while to load the metric to alarm and please refresh the alarm page.
3. The Metric Name is not properly configured while creating cloud watch resource. Please note that One Cloud watch Alarm can only monitor only one Metric.
4. There is no cost for the current month. i.e., The cost value is 0 therefore the Metric data becomes zero.

#### **6.6 Issue:** Incorrect Metric Data:

Please make sure that the Json files containing the cost data are up to date.



## Chapter 7 FAQs (Frequently Asked Questions)

Q1: What is the purpose of the notifier.py script?

Answer: The notifier.py script is a Python script that serves as the logic for the AWS Lambda function. It extracts information from CloudWatch Alarm notifications, such as alarm details and thresholds, and sends notifications through email and Slack.

Q2: How do I customize the email and Slack notifications?

Answer: You can customize the content and formatting of email and Slack notifications by modifying the respective sections in the notifier.py script. Update the HTML email body or Slack message text according to your preferences.

Q3: Can I modify the notification triggers or actions?

Answer: Yes, you can customize the notification triggers and actions by modifying the CloudWatch Alarm configurations that trigger the Lambda function. Additionally, you can extend the notifier.py script to perform additional actions based on your requirements.

Q4: How do I handle security for sensitive information?

Answer: Handle sensitive information, such as email addresses and Slack URLs, securely. Consider using AWS Secrets Manager or Parameter Store to store sensitive data and retrieve it securely within your Lambda function.

Q5: What if I encounter errors during Terraform execution?

Answer: Refer to the "Troubleshooting" section in this manual for guidance on addressing common issues. Check your Terraform command syntax, IAM role permissions, and resource configurations.

Q6: Can I deploy the Lambda function to a different AWS region?

Answer: Yes, you can modify the region variable in the variables.tf file to specify the desired AWS region for deployment.

## Chapter 8 Contact Information

If you have any questions, feedback, or need further assistance with setting up and using the CloudWatch Alarm notification mechanism using Terraform, you can reach out to us through the following channels:

- Email: For inquiries and communication, you can contact us via email at [swinabs@gmail.com](mailto:swinabs@gmail.com).
- GitHub: You can explore the code, report issues, or contribute to the project on GitHub by visiting our repository: <https://github.com/swinabs>.
- Slack: Join our Slack community to connect with fellow users, share experiences, and get help from experts. Join our Slack channel at <https://slack.example.com>.

We are committed to providing you with the support and assistance you need to successfully implement and manage CloudWatch Alarm notifications using Terraform. Feel free to contact us at any time.

## References

- [1] YouTube, "How to format the given link in IEEE standard," in YouTube. Available: <https://www.youtube.com/watch?v=oDcqqS73NXw>
- [2] XgridInc, "xc3," in GitHub. Available: <https://github.com/XgridInc/xc3>
- [3] Xgrid, "Unleashing the Power of Open Source: Introducing XC3 Cloud Cost Control," in Xgrid.co. Available: <https://www.xgrid.co/resources/unleashing-the-power-of-open-source-introducing-xc3-cloud-cost-control/>