

COS80029-Technology Application Project

Group User Manual

Team Members	Student ID
Zahin Akram	104100700
Abhishek Bikram KC	104057262
Sagar Poudel	104173061
Aamod Paudel	104184597

Contents

Group User Manual	1
1 Introduction.	4
1.1 Overview.	4
1.2 Purpose.	5
1.3 Scope.	5
1.4 Intended Audience.	5
1.5 Architecture of Implementation.	6
1.6 Warning.	6
2 Getting Started.	7
System Requirements	7
Installation	7
Installation of Pre-requisites	7
Packaging the Prometheus	7
2.1 Set-up Considerations.	8
2.2 User Access Considerations.	8
2.3 Accessing the System.	9
2.4 System Organization and Navigation.	9
2.5 Exiting the System.	9
2.6 Summary of Usage.	9
3. IAM Role Cost Lambda	10
3.1 IAM Role Cost Lambda	10
4 Integration.	11
4.1 Integrating python and terraform files.	11
4.2 Implementing Global Variables.	11
5. Troubleshooting.	16
5.1 Common Issues and Solutions.	16
5.1.1 Email Not Being Sent.	16
5.1.2 Slack Message Not Being.	16
5.1.3 List of IAM users Denied.	16
5.1.4 Issues with Terraform.	17
5.2 FAQs.	18
6. Appendices.	20

6.1 Sample Screenshots.....	20
Example of Email	20
Example of Slack Message.	21
Example of Grafana.....	21
6.2 Code Snippets.	22
Code for Email.....	22
Code for Slack.....	23
Code for Pushing metrics to Prometheus Push Gateway.....	25

1 Introduction.

1.1 Overview.

In today's cloud-dependent world, managing cloud costs effectively is a must. XC3, an open-source cloud cost control tool developed by XGrid, offers a powerful solution to this challenge.

XC3 empowers businesses to optimize their cloud infrastructure for cost savings. With real-time visibility into cloud usage and spending across multiple providers and accounts, businesses can identify and eliminate waste. Unused or underutilized resources are easily spotted, preventing unnecessary spending. Additionally, insightful dashboards and customizable alerts based on data analysis help businesses make informed decisions about resource allocation.

The strength of XC3 lies in its open-source foundation. It leverages powerful open-source tools like Cloud Custodian, Prometheus, and Grafana. Cloud Custodian enforces cost-saving policies and governance across cloud platforms. Prometheus collects and stores cloud cost metrics for analysis, while Grafana transforms this data into clear and interactive dashboards.

This open-source approach fosters collaboration. Users can contribute improvements and report issues, shaping XC3 into the go-to solution for cloud cost control.

Traditional cloud cost management struggles with complexities like cost attribution, identifying expensive resources, managing unused resources, limited visibility into cloud usage, and a lack of in-house expertise. XC3 addresses these challenges head-on.

XC3 offers a comprehensive suite of features including real-time cost and usage monitoring, idle resource detection, and customizable cost alerts. By implementing XC3, businesses can significantly reduce unnecessary cloud costs, prevent overspending, and maximize the return on their cloud investments. In essence, XC3 equips organizations with the tools and insights needed to effectively manage and control their cloud infrastructure expenses.

In this project, we aimed to expand the functionalities of the Identity and Access Management (IAM) Role Cost Breakdown feature within XC3, a cloud cost control tool designed specifically for AWS. This feature offers valuable insights into how costs are allocated based on specific IAM roles.

While the current implementation focuses on retrieving cost metrics for EC2 instances tied to IAM roles, our project centered on extending this capability to encompass

additional AWS services. We targeted services where costs can be directly attributed to IAM roles, including Lambda, SNS, and CloudWatch. It's important to acknowledge that this role-based cost breakdown approach isn't applicable to all services – for instance, S3 buckets don't utilize IAM roles. Through this expansion project for Lambda, SNS, and CloudWatch, we hope to empower users with a more granular understanding of how IAM roles influence costs across their AWS infrastructure.

1.2 Purpose.

This user manual accompanies the enhanced IAM Role Cost Breakdown feature for AWS within a cloud cost control tool. This feature provides a more detailed breakdown of costs associated with specific IAM roles. We expanded functionality beyond EC2 instances to include Lambda, SNS, and CloudWatch, where costs can be directly tied to IAM roles. This manual will guide you on leveraging this feature to gain a clearer understanding of how IAM roles influence costs across your AWS services. This knowledge will empower you to optimize your cloud spending through informed resource allocation decisions.

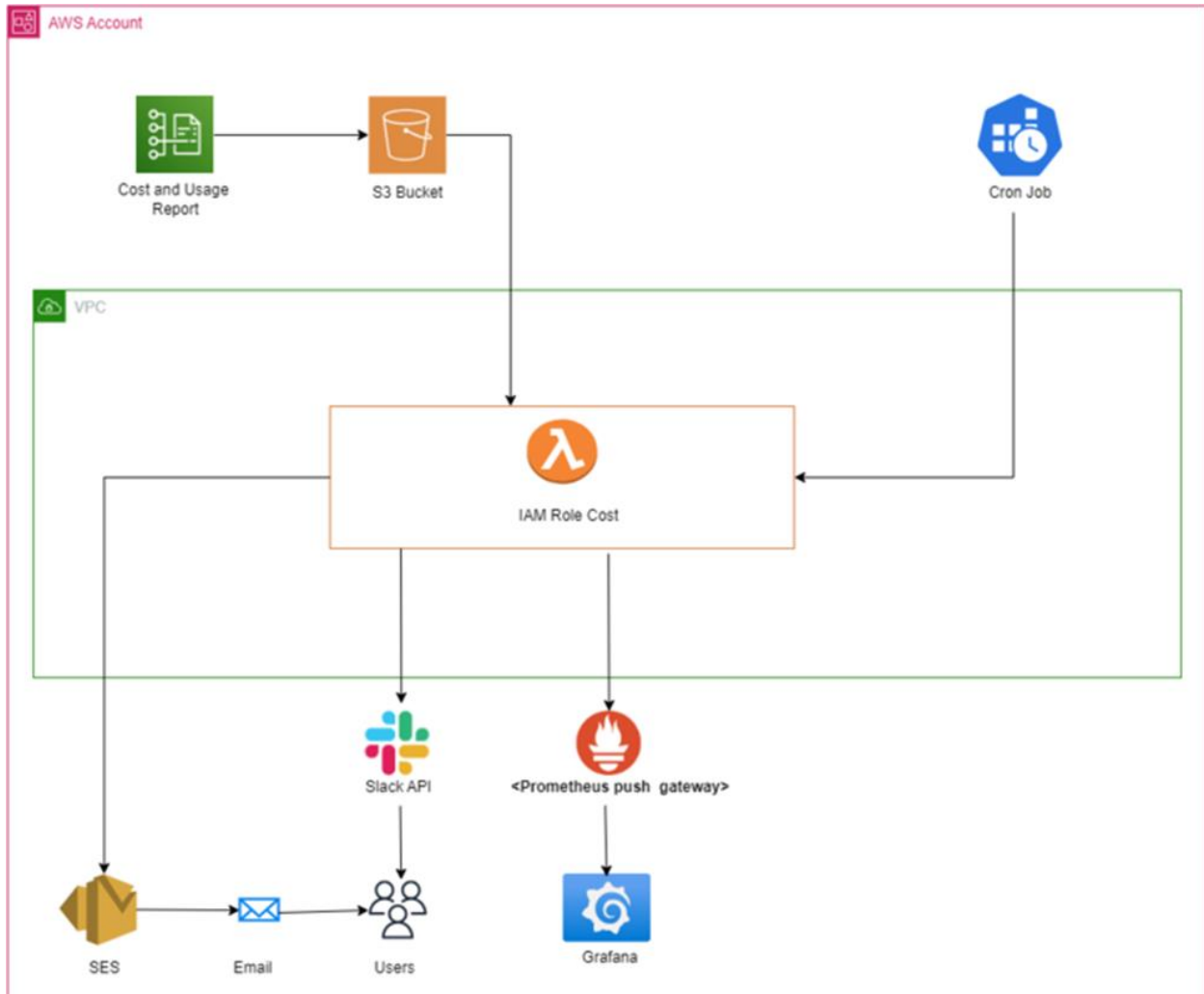
1.3 Scope.

This user manual focuses on utilizing the enhanced IAM Role Cost Breakdown feature within the XC3 cloud cost control tool for AWS. It guides users through leveraging this functionality for specific services (Lambda, SNS, and CloudWatch) where costs can be tied to IAM roles

1.4 Intended Audience.

This user manual serves as a technical guide for XC3 project members, instructors, and developers. Assuming familiarity with cloud technologies, AWS, Git version control, and Terraform deployments, the manual dives into the design, implementation, and integration of IAM Role Cost Breakdown within XC3, offering valuable insights for cost monitoring and reporting functionalities.

1.5 Architecture of Implementation.



This document details our proposed architecture for cost monitoring and reporting in XC3. A single Lambda function will be responsible for identifying resources across various services, calculating their costs, and pushing that data to Grafana for visualization via Prometheus Gateway. This approach leverages Cron jobs, similar to existing XC3 Lambda functions, to automate the Lambda execution at specific times or intervals. This ensures automatic resource identification, cost calculation, and visualization in Grafana.

1.6 Warning.

Using XC3 and its IAM-based Cost Breakdown requires providing your AWS account ID and security keys, which are essential for access but can be risky if shared. Disclose them only to trusted individuals when modifying or sharing related files, as a compromised account ID could lead to data leaks, unauthorized access, and unexpected charges.

2 Getting Started.

System Requirements

- Amazon Web Services (AWS) account.
- Installation of Pre-requirements
- Windows ZIP
- Powershell and Bash

Installation

Installation of Pre-requisites

- Terraform (Version 1.4.6) should be Installed, if not then you can install using: [Terraform Link](#)
- Python (Version 3.9) should be Installed, if not then you can install using : [Python Link](#)
- AWS CLI (Version 2) should be Installed, if not then you can install using : [AWS CLI Link](#)
- Configure AWS CLI (Your terminal should be able to access AWS Secret Key and Access Key) : [AWS CLI Configuration](#)
- The user has to enable the CostExplorer API in their AWS Account by using the following link: [AWS Cost Explorer](#)

Packaging the Prometheus

```
cd xc3/infrastructure/  
  
mkdir python  
  
pip3 install -t python/ prometheus-client  
  
zip -r python.zip ./python
```

Deployment

Step 1: After packaging the Prometheus, go to xc3/pre_requirement path and update the variables in the terraform.auto.tfvars file

Step 2: After modifying the variables, run the below commands:

Step 3: Now go to xc3/infrastructure/ path and update the variables in config.sh file

Step 4: After performing the above, run the commands given below:

Step 5: Configure the bucket in the backend.tf file in xc3/infrastructure. The bucket names should be same as in config.sh file.

Step 6: In xc3/infrastructure, configure the terraform.auto.tfvars file. Our function which is Cost and USage Report needs to be enabled and its path must be provided in the terraform.auto.tfvars since it retrieves costs from the Cost and Usage Report (CUR). It is necessary to define variables such as CUR_s3_file_key, which is the file key containing the cost and usage report, and CUR_s3_bucket_name, which is the bucket name. Additionally, slack_channel_url,.... should be defined for the added feature of receiving notifications on Slack.

Step 7: After modifying the above and changing namespace, project, region and with email addresses in terraform.auto.tfvars file. Now run the commands given below:

Enter “yes” to apply the infrastructure. Then check the lb dns to verify that the infrastructure is deployed successfully.

2.1 Set-up Considerations.

Equipment: This feature requires a modern web browser, an Integrated Development Environment (IDE) for coding, and a computer or device with internet access. Additionally, you'll need the ability to access and utilize AWS services.

System Configuration: To implement this feature, you'll need a modern web browser, an Integrated Development Environment (IDE) for coding, and a computer or device with internet access. Additionally, you'll require access to AWS services and the ability to generate and analyze Cost and Usage reports to monitor resource utilization and potential costs associated with the feature.

2.2 User Access Considerations.

User Roles: For secure operation, this feature requires an IAM user created within your AWS account. This user should have specific permissions assigned, following the principle of least

privilege. This ensures only necessary actions can be performed and prevents unauthorized access to sensitive resources.

2.3 Accessing the System.

Accessing the system is a breeze! Simply navigate to the AWS Management Console (link to AWS Management Console: <https://aws.amazon.com/console/>) and sign in using your existing AWS account credentials. This leverages your existing permissions, making it quick and convenient to get started.

2.4 System Organization and Navigation.

Organization: The IAM Role based cost breakdown feature is integrated within the XC3 project environment. The main components include Lambda functions, Cloudwatch, Cost and Usage Report, SNS .

Navigation: After accessing the AWS account, navigate to the relevant services and functions associated with the IAM Role based cost breakdown feature. For specific instructions, refer to the corresponding sub-sections below.

2.5 Exiting the System.

Before signing off, remember to ensure a clean exit. First, navigate to the infrastructure folder and run terraform destroy to safely remove the resources you deployed. To avoid unnecessary storage charges, remember to manually empty the metadata-storage bucket in S3 before destruction. Finally, log out of your AWS account and close any open browser sessions for security.

2.6 Summary of Usage.

Our features seamlessly integrate into your existing XC3 setup. Follow the standard XC3 installation process, or refer to the provided installation guide for alternative methods. The only additional step required is obtaining your Slack webhook URL (instructions included above). All other configurations, including our custom variables (which simply complement existing options), are handled automatically during deployment. This allows you to leverage our functionalities with minimal effort.

3. IAM Role Cost Lambda

3.1 IAM Role Cost Lambda

Our Lambda function fetches AWS cost from Cost and Usage Report from S3 bucket, process the data to map costs to IAM roles based on Lambda functions, SNS and CloudWatch. The cost are then pushed to Grafana via Prometheus Pushgateway. For additional functionality, we have also implemented notifications systems where we send email with the cost breakdown using Amazon SES along with Slack notifications to Slack channel using webhook.

The **fetch_lambda_to_role_mapping(lambda_client)** function fetches the mapping from Lambda function ARNs to their associated IAM Role ARNs. It uses the `list_functions` API to retrieve Lambda functions and their roles. At last, it returns two dictionaries which is one mapping Lambda ARNs to role ARNs and another mapping Lambda names to role ARNs.

The given snippet, **fetch_topic_subscriptions(sns_client, topic_arns)** function fetches the subscriptions for the given SNS topic ARNs. It uses `list_subscriptions_by_topic` API to retrieve subscriptions for each topic. It then, returns a dictionary mapping topic ARNs to a list of Lambda endpoint ARNs subscribed to that topic.

The **fetch_cw_to_role_mapping(lambda_name_to_role_arn_mapping, cw_log_group_arns)** function maps IAM role ARNs for the given CloudWatch log group ARNs. It extracts the Lambda function name from the CloudWatch log group ARN and looks up the corresponding IAM role ARN. Then it returns a dictionary mapping CloudWatch log group ARNs to their associated IAM role ARNs.

The **map_costs_to_roles(csv_rows, lambda_to_role_mapping, topic_subscriptions, name_to_role_mapping, cw_to_role_mapping)** function maps costs to IAM roles based on Lambda functions, SNS subscriptions and CloudWatch logs. It iterates over the CSV rows and checks the product code to determine the cost category (Lambda, SNS and CloudWatch). It also uses respective mappings to associate costs with IAM roles. Then it returns a dictionary mapping IAM role ARNs to their costs broken down by category.

The **push_metrics_to_pushgateway(role_costs, prometheus_pushgateway)** function pushes the mapped costs by IAM roles to Prometheus Pushgateway. It creates Prometheus Gauges for total cost, Lambda cost, SNS cost and CloudWatch cost by IAM Role. It sets the gauge values based on the cost data. Then it pushes the metrics to the specified Prometheus Pushgateway.

The function **send_email_ses(role_costs, sender_email, recipient_email)** sends an email with the cost breakdown using Amazon SES. It constructs the email subject and body based on the cost data. It uses SES client to send the email from the sender to the recipient.

The **send_to_slack(payload, webhook_url)** function sends the cost breakdown payload to a Slack channel using webhook. It converts the payload to JSON and sends it as an HTTP POST request to the specified webhook URL.

The **lambda_handler(event, context)** function is the main Lambda function handler. It retrieves the necessary environment variables. It initializes the required AWS clients which is S3, Lambda and SNS. It calls the respective functions to download and parse the CSV file, fetch mappings, map costs to roles and push metrics to Grafana via Prometheus Pushgateway. Then it sends an email with the cost breakdown using SES and posts the cost breakdown to a Slack channel.

4 Integration.

4.1 Integrating python and terraform files.

Python scripts and Terraform files must cooperate flawlessly for XC3 development to proceed smoothly. By combining Terraform's infrastructure management with Python's scripting capabilities, our partnership produces a development approach that is both simplified and effective.

1. Initially, a thorough examination of the contents is conducted in order to comprehend the features' needs and functionality.
2. To guarantee file consistency, the Python file must then be thoughtfully positioned inside the **'/src'** subdirectory.
3. The **'/infrastructure/modules/serverless/'** subdirectory contains the Terraform. Accessibility is further ensured by grouping related files together.

Note: Verify that the names of the terraform and python files accurately represent their functions.

4.2 Implementing Global Variables.

Throughout the project, global variables are essential to preserving modularity and consistency. They enable us to describe important settings, data, and parameters just once, guaranteeing that updates or modifications are reflected consistently throughout the system. This encourages code reuse, cuts down on redundancy, and makes maintenance easier.

Method and Implementation: A structured approach was developed, integrated, and the current code was thoroughly analyzed in order to implement global variables in an efficient manner. Because it was necessary to measure the process's efficiency, the implementation was done gradually.

1. Defining global values

Locate the file "terraform.auto.tfvars" in the xc3/infrastructure folder. Give the variable a name and the desired value. Global variables are all those defined in terraform.auto.tfvars.

```
Team01 / infrastructure / terraform.auto.tfvars

Code Blame 82 lines (77 loc) · 2.9 KB

14
15 namespace = "xc3sagarpoudel"
16 env = "dev"
17 region = "ap-southeast-2"
18 account_id = "211125640160"
19 vpc_cidr_block = "10.0.0.0/16"
20 public_subnet_cidr_block = {
21   "ap-southeast-2a" = "10.0.0.0/24"
22   "ap-southeast-2b" = "10.0.1.0/24"
23 }
24 domain_name = ""
25 hosted_zone_id = "Z053166920YP15TI0EK5X"
26
27 private_subnet_cidr_block = {
28   "ap-southeast-2a" = "10.0.100.0/24"
29 }
30 # private_subnet_cidr_block = "10.0.100.0/24"
31 allow_traffic = ["0.0.0.0/0"] // Use your own network CIDR
32 ses_email_address = "mailtosagarpoudel@gmail.com"
33 receiver_email_address = "mailtosagarpoudel@gmail.com"
34 creator_email = "mailtosagarpoudel@gmail.com"
35 owner_email = "mailtosagarpoudel@gmail.com"
36 instance_type = "t2.micro"
37 total_account_cost_lambda = "total_account_cost"
38 iam_role_cost_lambda = "iam_role_cost"
39 CUR_s3_bucket_name = "team1reportbucket"
40 CUR_s3_file_key = "report/mycostreport/20240401-20240501/20240405T101631Z/mycostreport-00002.csv"
41 slack_channel_url = "https://hooks.slack.com/services/0011T06FJLP725D0011/0011B06TQHGH5Q0011/0011Rqmhc6wxZ1tLx1NWXf1EtI10011" #omit 0011s
42 slack_channel = "C06NAMZR69E"
43 slack_icon_emoji = ":rocket:"
44 slack_username = "bot"
45 total_account_cost_cronjob = "cron(0 0 1,15 * ? *)" // flexible can be set according to need
46 prometheus_layer = "lambda_layers/python.zip" // s3 key for lambda layer
47 memory_size = 128
48 timeout = 300
49 project = "xc3sagarpoudel"
50 create_cloudtrail_kms = false
51 create_cloudtrail = false
52 create_cloudtrail_s3_bucket = false
```

a. Now, go to variables.tf file under xc3/infrastructure and define the variable. For example:

```
variable "CUR_s3_bucket_name"{
  description = "bucket name where CUR is located"
  type       = string
}

variable "CUR_s3_file_key"{
  description = "path of csv file where cur is located"
  type       = string
}

variable "slack_channel_url"{
  description = "slack web hook url to send notification via slack"
  type       = string
}

variable "slack_username"{
```

2. Using the global variables in terraform

Once the global variables are defined, they can be called by the terraform files. The syntax for calling is 'var.{variable name}'.

For example, in iam_role_cost.tf, namespace can be used like:

```
5 }
6
7 # Creating IAM Role for Lambda functions
8 resource "aws_iam_role" "iam_role_cost" {
9   name = "${var.namespace}-${var.iam_role_cost_lambda}-role"
10  assume_role_policy = jsonencode({
11    Version = "2012-10-17"
12    Statement = [
13      {
14        Action = "sts:AssumeRole"
15        Effect = "Allow"
16        Sid    = "IAMRoleSNSFunction"
17        Principal = {
18          Service = "lambda.amazonaws.com"
19        }
20      }
21    ]
22  })
23 }
```

3. Using the global variables in python file

The AWS Lambda function configuration defined in the Terraform file enables you to set environment variables that are accessible from the related Python code. With this functionality, you may safely transfer configuration settings and dynamic data from Terraform to your Python code in a smooth and efficient manner.

```

resource "aws_lambda_function" "iam_role_cost" {
  function_name = "${var.namespace}-${var.iam_role_cost_lambda}"
  role          = aws_iam_role.iam_role_cost.arn
  runtime       = "python3.9"
  handler       = "${var.iam_role_cost_lambda}.lambda_handler"
  filename      = data.archive_file.iam_role_cost.output_path
  environment {
    variables = {
      prometheus_ip = "${var.prometheus_ip}:9091"
      account_detail = var.namespace
      slack_channel_url = var.slack_channel_url
      slack_channel     = var.slack_channel
      slack_icon_emoji  = var.slack_icon_emoji
      slack_username    = var.slack_username
      ses_email_address = var.ses_email_address
      receiver_email_address = var.receiver_email_address

      CUR_s3_bucket_name = var.CUR_s3_bucket_name
      CUR_s3_file_key    = var.CUR_s3_file_key
    }
  }
  memory_size = var.memory_size
  timeout     = var.timeout
  layers      = [var.prometheus_layer]
  vpc_config {
    subnet_ids      = [var.subnet_id[0]]
    security_group_ids = [var.security_group_id]
  }
  tags = merge(local.tags, tomap({ "Name" = "${var.namespace}-iam_role_cost" }))
}

```

In the Terraform code snippet provided, you can see that we are setting multiple environment variables within the `aws_lambda_function` resource block. These variables, such as `CUR_s3_file_key`, `slack_channel_url`, `ses_email_address`, and `CUR_s3_bucket_name`, are critical parameters required for the operation of the Lambda function.

The `os` module can now be used to retrieve these environment variables within your Python code. This is an illustration of how to utilize and access these variables in your Python code:

```
def lambda_handler(event, context):
    bucket_name = os.environ.get("CUR_s3_bucket_name")
    file_key = os.environ.get("CUR_s3_file_key")
```

You can retrieve the values of the environment variables you created in the Terraform file by using the `os.environ[""]` function. This method makes sure that dynamic data, configuration parameters, and sensitive information are securely sent to your Python code during runtime.

Note: In terraform files, make sure you provide dynamic names to the resources to avoid conflict with pre-existing resources.

```
# Creating IAM Role for Lambda functions
resource "aws_iam_role" "lambda_execution_role_IamRolestoGrafana" {
    name = "${var.namespace}-iamrolestografana"
    assume_role_policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Action = "sts:AssumeRole"
                Effect = "Allow"
                Sid    = "iamrolestografanarole"
            }
        ]
    })
}
```

```
}

# Creating IAM Role for Lambda functions
resource "aws_iam_role" "iam_role_cost" {
    name = "${var.namespace}-${var.iam_role_cost_lambda}-role"
    assume_role_policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Action = "sts:AssumeRole"
                Effect = "Allow"
                Sid    = "IAMRoleSNSFunction"
            }
        ]
    })
}
```

For instance, the resource `"aws_iam_role"` in the snippet above was created in two separate terraform files, but its names—`"iam_role"` and `"iam_role_cost"`, which are specific to that one terraform file—are different. Additionally, by placing the namespace before the actual name, the resource's name is rendered dynamic. By doing this, it is ensured that there are no conflicts with

the previously created resources. It's a good idea to give redundant resources distinct names for convenience's sake when they're created more than once.

5. Troubleshooting.

5.1 Common Issues and Solutions.

5.1.1 Email Not Being Sent.

One possible problem for users is that they aren't getting emails. There are several ways to address this:

- Check the spelling of the source and destination email addresses.
- Check that the source email address is confirmed. The source email address receives an email for authentication and verification when the email is scheduled to be delivered.
- Check the SPAM folder to ensure that emails are not automatically redirected there.
- If the SNS/SES policy isn't connected, there may be an error in the SNS/SES arn.

5.1.2 Slack Message Not Being.

If you're having trouble sending messages to Slack, try these fixes:

- Verify the Slack API that the user established to make sure that permissions are granted appropriately.
- Verify that the policies are correctly attached.

5.1.3 List of IAM users Denied.

If you're encountering "IAM User Denied" errors while trying to execute AWS Lambda functions, consider the following troubleshooting steps:

- Check the IAM policies that are linked to the role or IAM user that is connected to the Lambda function. Make sure the policies are set up properly to permit the necessary operations for the Lambda function.
- You'll need to create more IAM policies to provide the necessary rights if some are lacking.

As an illustration:

```
{  
  "Sid": "ListIAMRoles",  
  "Effect": "Allow",  
  "Action": [
```



```
    "iam:ListRoles"
  ],
  "Resource": "*"
}
```

- Verify that the Resource attribute has the right AWS resources and that the policy syntax is valid. Permission problems may arise from improper resource definitions or syntax.

5.1.4 Issues with Terraform.

Here's a troubleshooting guide for potential issues that might arise while working with the Terraform code.

IAM Role and Policy Configuration:

Issue: The IAM role or policies aren't created properly.

Troubleshooting: Double-check the IAM role and policy resources. Verify their names, JSON policy documents, and their associations. Make sure there are no typos or syntax errors.

Lambda function Setup:

Issue: Lambda function doesn't execute as expected.

Troubleshooting: Ensure the function's role, handler, runtime, filename, and timeout are correctly set. Verify that the function's ZIP file is generated and uploaded successfully.

CloudWatch Events Rule:

Issue: Lambda function isn't triggered by CloudWatch Events.

Troubleshooting: Review the CloudWatch Events rule configuration. Confirm that the schedule expression is valid. Check for any overlapping rules that might affect triggering.

Lambda Function Permission:

Issue: Permission issues causing CloudWatch Events not to invoke the Lambda.

Troubleshooting: Validate the "eventbridge_permission" resource. Ensure the Lambda function's ARN and the CloudWatch Events rule's ARN are accurate. Verify that the principal is set to "events.amazonaws.com".

AWS Provider Configuration:

Issue: Resources fail to create due to incorrect AWS provider configuration.

Troubleshooting: Check the AWS provider configuration at the beginning of the code. Verify the specified region and other settings are accurate.

General Errors:

Issue: Resources fail to create due to various errors.

Troubleshooting: Examine the error messages returned by Terraform. These messages can provide insights into what went wrong. Review your code and make sure there are no missing or extra attributes, and all resource dependencies are properly defined.

Permissions and Policies:

Issue: Resources are created, but they don't behave as expected due to inadequate permissions.

Troubleshooting: Verify that the IAM roles and policies grant the necessary permissions to access required resources. Cross-check your permissions settings with the intended actions of each resource.

Resource Naming Conflicts:

Issue: Resources fail to create due to naming conflicts.

Troubleshooting: Ensure that resource names, such as IAM roles, policies, and Lambda functions, are unique within your AWS account and adhere to AWS naming conventions.

Terraform State Management:

Issue: Inconsistent state between local configurations and remote state.

Troubleshooting: Regularly run terraform init and terraform plan to refresh your local state with the remote state. If issues persist, consider using remote backends for state storage.

5.2 FAQs.

Q1: What is the frequency of the IAM Role Cost Lambda's cost breakdown updates?

The IAM Role Cost Lambda updates the cost breakdown on a regular basis. By default, it may be set to update daily. However, users can customize this frequency according to their requirements by adjusting the configuration settings.

Q2: Is it possible to customize the notification messages sent by the IAM Role Cost Lambda to Slack and email?

Yes, users have the flexibility to personalize the content of notification messages. They can include relevant details and context in the messages to suit their needs. Consult the documentation for instructions on modifying notification templates.

Q3: How does the IAM Role Cost Lambda handle errors during data processing?

If an error occurs during data processing, the IAM Role Cost Lambda logs the specifics of the error. Users can refer to the Lambda function logs to identify the root cause of the error and take necessary corrective actions to resolve it.

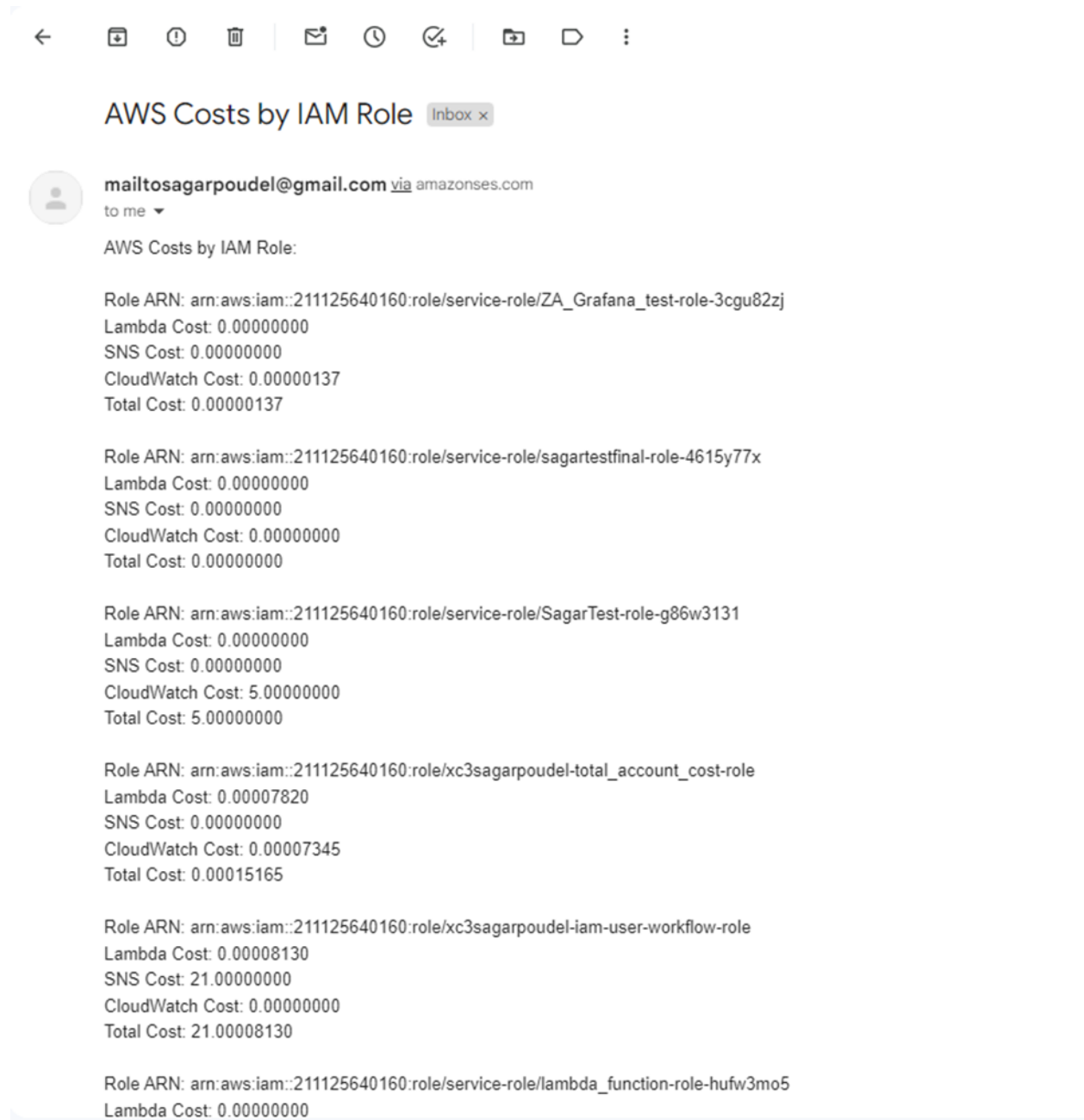
Q4: Is there a limit to the number of Slack channels and email addresses that can receive notifications?

There isn't a strict cap on the number of recipients. However, users should consider best practices and their preferred communication methods to ensure that notifications are effectively delivered to relevant parties.


6. Appendices.

6.1 Sample Screenshots.

Example of Email



Example of Slack Message.

**NotificationTestingIamRole** APP 1:20 PM

Today ▾

AWS Costs by IAM Role:

Role ARN: arn:aws:iam::211125640160:role/service-role/ZA_Grafana_test-role-3cgu82zj
Lambda Cost: 0.00000000
SNS Cost: 0.00000000
CloudWatch Cost: 0.00000137
Total Cost: 0.00000137

Role ARN: arn:aws:iam::211125640160:role/service-role/sagartestfinal-role-4615y77x
Lambda Cost: 0.00000000
SNS Cost: 0.00000000
CloudWatch Cost: 0.00000000
Total Cost: 0.00000000

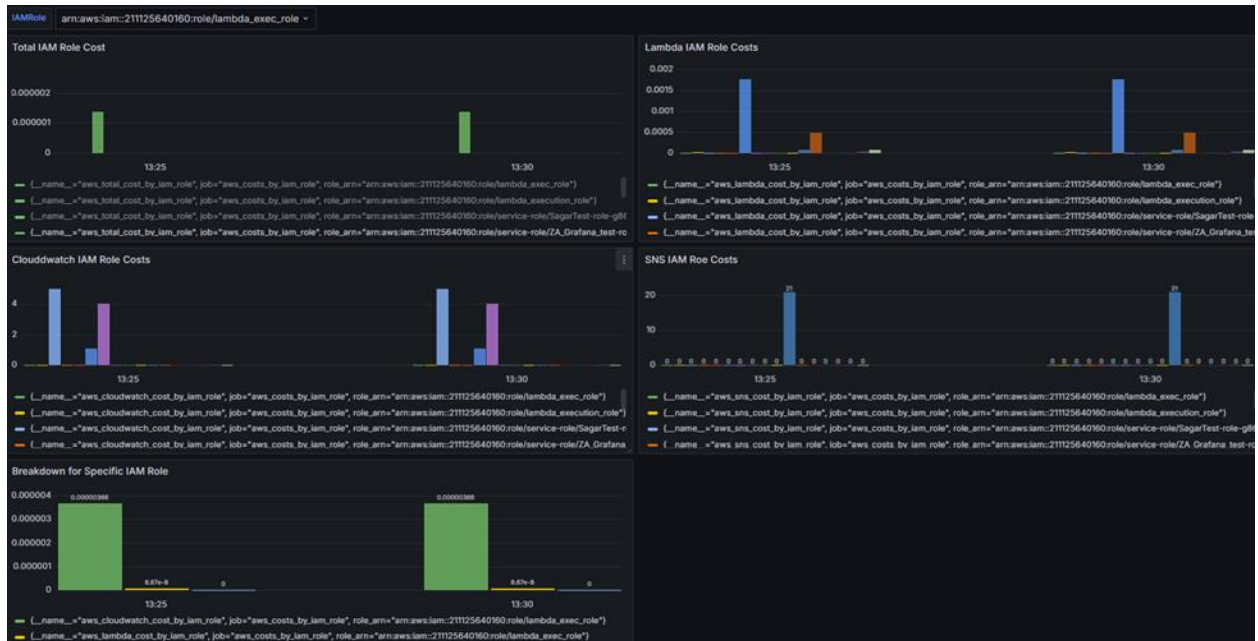
Role ARN: arn:aws:iam::211125640160:role/service-role/SagarTest-role-g86w3131
Lambda Cost: 0.00000000
SNS Cost: 0.00000000
CloudWatch Cost: 5.00000000
Total Cost: 5.00000000

Role ARN: arn:aws:iam::211125640160:role/xc3sagarpoudel-total_account_cost-role
Lambda Cost: 0.00007820
SNS Cost: 0.00000000
CloudWatch Cost: 0.00007345
Total Cost: 0.00015165

Role ARN: arn:aws:iam::211125640160:role/xc3sagarpoudel-iam-user-workflow-role
Lambda Cost: 0.00008130
SNS Cost: 21.00000000
CloudWatch Cost: 0.00000000
Total Cost: 21.00008130

Role ARN: arn:aws:iam::211125640160:role/service-role/lambda_function-role-hufw3mo5
Lambda Cost: 0.00000000
SNS Cost: 0.00000000
CloudWatch Cost: 0.00000243
Total Cost: 0.00000243

Example of Grafana



6.2 Code Snippets.

Code for Email

The simplest code for sending an email message is the one that follows. This message is extremely straightforward. Since this is the simplest message that is being sent. A sample of the code is given below:

```

def send_email_ses(role_costs, sender_email, recipient_email):
    # Create the email message
    subject = "AWS Costs by IAM Role"
    body = "AWS Costs by IAM Role:\n\n"
    for role_arn, costs in role_costs.items():
        body += f"Role ARN: {role_arn}\n"
        body += f"Lambda Cost: {costs.get('LambdaCost', 0):.8f}\n"
        body += f"SNS Cost: {costs.get('SNSCost', 0):.8f}\n"
        body += f"CloudWatch Cost: {costs.get('CloudWatchCost', 0):.8f}\n"
        body += f"Total Cost: {sum(costs.values()):.8f}\n\n"

    # Create an SES client
    ses_client = boto3.client("ses")

    # Send the email using SES
    response = ses_client.send_email(
        Source=sender_email,
        Destination={"ToAddresses": [recipient_email]},
        Message={
            "Subject": {"Data": subject},
            "Body": {"Text": {"Data": body}},
        },
    )
    print(f"Email sent! Message ID: {response['MessageId']}")
    return body

```

Code for Slack.

The code for a basic Slack message is as follows. It uses the webhook url generated (guideline for generating it is provided).

```
body = send_email_ses(role_costs, sender_email, recipient_email)
```

```
# Format the payload for Slack
```

```
payload = {  
    'text': body,  
    'channel': slack_channel,  
    'username': slack_username,  
    'icon_emoji': slack_icon_emoji  
}
```

```
# Send the payload to Slack via webhook
```

```
send_to_slack(payload, slack_channel_url)
```

```
def send_to_slack(payload, webhook_url):
```

```
    # Convert the payload to JSON
```

```
    data = json.dumps(payload).encode('utf-8')
```

```
    # Send the payload to Slack
```

```
    req = urllib.request.Request(webhook_url, data=data, headers={'Content-Type': 'application/json'})
```

```
    urllib.request.urlopen(req)
```


Code for Pushing metrics to Prometheus Push Gateway

```
def push_metrics_to_pushgateway(role_costs, prometheus_pushgateway):
    """Push mapped costs by IAM roles to Prometheus Pushgateway."""
    registry = CollectorRegistry()

    # Gauge for Total costs by IAM role
    total_cost_gauge = Gauge(
        "aws_total_cost_by_iam_role",
        "Total Cost by IAM Role",
        ["role_arn"],
        registry=registry,
    )

    # Gauge for Lambda costs by IAM role
    lambda_cost_gauge = Gauge(
        "aws_lambda_cost_by_iam_role",
        "Cost of AWS Lambda by IAM Role",
        ["role_arn"],
        registry=registry,
    )

    # Gauge for SNS costs by IAM role
    sns_cost_gauge = Gauge(
        "aws_sns_cost_by_iam_role",
        "Cost of Amazon SNS by IAM Role",
        ["role_arn"],
        registry=registry,
    )

    # Gauge for CloudWatch costs by IAM role
    cloudwatch_cost_gauge = Gauge(
        "aws_cloudwatch_cost_by_iam_role",
        "Cost of Amazon CloudWatch by IAM Role",
        ["role_arn"],
        registry=registry,
    )

    for role_arn, costs in role_costs.items():
        # Calculate the total cost per IAM role
        total_cost = sum(costs.values())
        total_cost_gauge.labels(role_arn=role_arn).set(total_cost)
        lambda_cost_gauge.labels(role_arn=role_arn).set(costs.get("LambdaCost", 0))
        sns_cost_gauge.labels(role_arn=role_arn).set(costs.get("SNSCost", 0))
        cloudwatch_cost_gauge.labels(role_arn=role_arn).set(
            costs.get("CloudWatchCost", 0)
        )

    push_to_gateway(
        prometheus_pushgateway, job="aws_costs_by_iam_role", registry=registry
    )
```

