



USER MANUAL FOR XC3

IAM ROLE WORKFLOW
ENHANCEMENT (FULL
DELIVERABLE)

TEAM 14

Group Member
Sparsh Mishra
Sufiyan Ahmed Khan
Veera Korla
Preety Nagpal
Manju Rani Dinch

TEAM 14

CONTENTS

PREFACE & TERMS OF USE.....	Error! Bookmark not defined.
CONTENTS.....	2
INTRODUCTION.....	3
THE OVERALL TOOL: XC3	3
USER REQUIREMTNS (Individual Responsibility).....	3
INDIVIDUAL RESPOINSIBILITIES.....	3
COMPONENT: LAMBDA FUNCTIONS AND AUTOMATION	4
INSTALLATION.....	4
PRE-REQUIREMENTS.....	4
CLONING THE REPOSITORY FOR XC3	5
CONFIGURING THE REPOSITORY	5
DEPLOTING THE TOOL.....	7
WHY THIS VERSION/ COMPONENT OF XC3	10
FROM THE DEVELOPER’S NOTEBOOK.....	10
COMPONENT FEATURES	10
HOW TO USE: TRIGGER AND OPERATE	10
MANUAL TRIGGER	11
VERIFY PROPOER WORKING	11
SETTING AUTOMATION PERIOD	13
DEBUGGING	14
FIX.....	14
FIX.....	14
COMPONENT: DASHBOARD	15
HOW TO ACCESS AND NAVIGATE THE PANEL	15
ACCESS THE DASHBOARD	15
USING THE DROP DOWNS.....	15
USING THE SERVICE DROP DOWN	16
IAM ROLE DETAILS	17
SERVICE COST PANEL <i>Fig: IAM Roles Detail panel</i>	17
COMPONENT/ CONTRIBUTION DISECTION	Error! Bookmark not defined.
LAMBDA 1 – ROLES LAMBDA	19
LAMBDA 2 –COST LAMBDA.....	20
AUTOMATION MECHANISM.....	21
TERRAFORM : RESOURCE DEFINITIONS AND CONFIG.....	22
COST CALCULATION RESOURCES (EC2, S3, LAMBDA, DYNAMO)	23
PAYLOAD MECHANISM	24

INTRODUCTION

THE OVERALL TOOL: XC3

XC3 is a cloud agnostic tool for resource monitoring and tagging compliance, along with additional monitoring features for cloud platforms. The tool allows users to keep track of their cloud deployments under accounts that the deployments have been made under. The main functionalities include:

- IAM User based service cost breakdown
- IAM Role-based resource cost breakdown
- Federated User resource cost breakdown
- Resource Tagging compliance

These are the features that make XC3 a one-stop solution for all cloud management problems by providing a management ecosystem. It uses an efficient and mainstream combination of Prometheus and Grafana to provide database and visualisation solutions for the users.

INDIVIDUAL FEATURE. COMPONENT WORKED ON

USER REQUIREMENTS (Individual Responsibility)

These are the client's (XGrid) User requirements that the component this user manual aims to describe has been developed and designed in accordance with:

- **Enhance Cost Breakdown Functionality:**
Revise existing code and incorporate AWS APIs to dissect costs by IAM roles in the XC3 platform. Validate accuracy through meticulous testing, considering resource usage and billing data.
- **Integrate IAM Role Costs into Grafana Dashboard:**
Seamlessly embed IAM role-based cost breakdown into XC3's Grafana dashboard, ensuring consistency in design and usability standards.

Additional Requirements

- **Test, Debug, and Ensure Compatibility:**
Conduct thorough testing and debugging across diverse environments and scenarios to guarantee seamless functionality and compatibility of the IAM role-based cost breakdown feature
- **Document and Share Knowledge:**
Craft comprehensive documentation elucidating the feature's functionality and usage to facilitate its adoption and comprehension among users and team members.
- **Review and Acceptance:**
Collaborate with the team to refine the feature based on feedback, ensuring its readiness for acceptance before finalizing the implementation.

INDIVIDUAL RESPONSIBILITIES

- Develop Lambda function to fetch all IAM Roles under an account
- Develop Lambda function to map services affiliated with the fetched IAM Roles
- Implement logic to fetch resources for the mapped services
- Implement functionality to fetch costs for resources fetched
- Polish logic and framework to make deliverable efficient and fault tolerance (Payload)
- Automate the workflow for users

COMPONENT: LAMBDA FUNCTIONS AND AUTOMATION

The following section outlines the steps necessary for installing and using the components that have been designed individually by the author (Sparsh Mishra) for the project in XC3 code base/ repository.

INSTALLATION

PRE-REQUIREMENTS

These are the necessary pre-requirements that the user must satisfy to be able to utilise the full strength of XC3 (specific component(s)).

MINIMUM REQUIREMENTS

- A Linux sub-system for installing and utilising Terraform
 - WSL For Windows, or
 - Linux Subsystem for macOS
- Terraform installed on the above Linux Subsystem (latest version)
- Python (v3.9.0 or above) installed on WSL
- AWS CLI (v2,0) or above installed
- Set up the AWS Command Line Interface (CLI) to enable access to your AWS account using your terminal by providing your AWS Access Key ID and Secret Access Key
- Enable the Cost Explorer API for the AWS account that the AWS CLI has been configured with

RECOMMENDED REQUIREMENTS

- Visual Studio Code ([latest version](#) for macOS or Windows) on local machine for managing WSL and repository hosted on it

After having satisfied these requirements, your machine is now ready to experience agnostic cloud monitoring expertise with xc3 (individual component(s) developed by Sparsh Mishra)

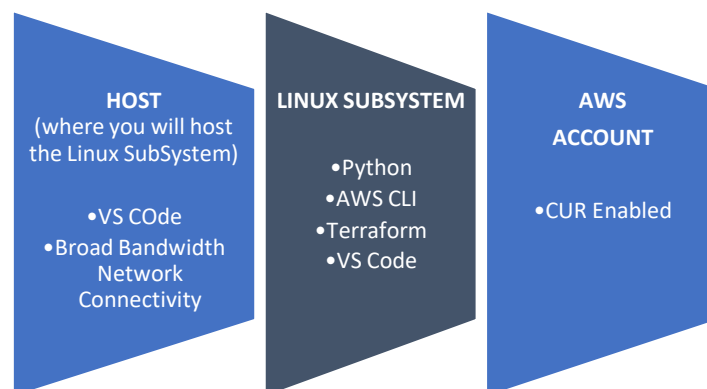


Figure 1 Technology Stack Ready for Installation

The above conditions are sufficient to enjoy all the features of the described XC3 components(individual components). The next step is to install the tool, by cloning the repository that contains all necessary files and dependencies on the Linux Subsystem and performing some configuration and tuning.

CLONING THE REPOSITORY FOR XC3

(Assuming the contribution from the team has been merged with the XC3 repository on GitHub)

Run this command from the Linux Subsystem terminal:

```
git clone https://github.com/XgridInc/xc3.git
```

OR

(Assuming the need to inspect the team branch))

Navigate to GitHub Team branch using :

```
https://github.com/104202994/xc3/tree/main
```

And then clone the repo on the WSL from the interface

After this step, your host Linux subsystem should have the XC3 repository cloned/ downloaded locally.

CONFIGURING THE REPOSITORY

Now, let us set up the repository and the tool personalised according to your AWS account. (Keep your details handy for this one)

PROMETHEUS CLIENT INSTALLATION

Navigate to your XC3 directory on your Linux Subsystem, and follow these commands:

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

Now, after the above commands finish executing, your Prometheus client is all set!, let us move on...

TUNING/ PERSONALISING THE REPOSITORY

- Navigate to the 'pre_requirement' directory (preferably using VS Code) and personalise the 'terraform.auto.tfvars' file according to your credential and project requirements:



```
project = "example" ←  
creator_email = "example@example.co" ←  
owner_email = "example@example.co" ←  
namespace = "example" ←  
region = "ap-southeast-2" ←
```

Figure 2 What to Change in pre_req/tfvars file

- Now, navigate to 'infrastructure' directory and update the 'variables.tf' file for your use case

```

export aws_region="ap-southeast-2" ←
export dynamo_table_name="terraform-lock" Dont change DynamoDb Table name
export bucket_name="terraform-state-team01" Please change according to your Team ←
export project="example" ←
export domain="testing.example.co" ←
export owner_email="admin@example.co" ←
export creator_email="admin@example.co" ←
export namespace="example" ←

```

Figure 3 What to change in infra/variables

- Next, still in 'infrastructure', make changes to backend.tf and config.sh files, ensuring the bucket names are consistent everywhere

```

15 // Terraform Remote State
16
17 terraform {
18   backend "s3" {
19     bucket = "example-state-bucket" // S3 bucket for terraform state management ←
20     key    = "xc3/xc3.tfstate"      // S3 object key for terraform state file to maintain history of resources
21     region = "ap-southeast-2"
22     dynamodb_table = "terraform-lock"
23   }
24 }
25

```

Figure 4 Changes in Backend.tf and config.sh files

- And lastly (we promise), in the 'infrastructure' directory, personalise the 'terraform.auto.tfvars' file

```

14 namespace = "example111" ←
15 env       = "dev" ←
16 region    = "eu-west-1" ←
17 account_id = "123456789012" ←
18 vpc_cidr_block = "10.0.0.0/16" ←
19 public_subnet_cidr_block = {
20   "eu-west-1a" = "10.0.0.0/24"
21   "eu-west-1c" = "10.0.1.0/24"
22 }
23 domain_name = "" ←
24 hosted_zone_id = "Z053166920YP1STI0E5X" ←
25
26 private_subnet_cidr_block = {
27   "eu-west-1a" = "10.0.100.0/24"
28 }
29 # private_subnet_cidr_block = "10.0.100.0/24"
30 allow_traffic = ["125.209.64.242/32"] // Use your own network CIDR ←
31 ses_email_address = "admin@example.co" ←
32 creator_email     = "admin@example.co" ←
33 owner_email       = "admin@example.co" ←
34 instance_type     = "t2.micro"
35 total_account_cost_lambda = "total_account_cost"
36 total_account_cost_cronjob = "cron(0 0 1,15 * ? *)" // Flexible can be set according to need
37 prometheus_layer = "lambda_layers/python.zip" // s3 key for lambda layer
38 memory_size      = 128
39 timeout          = 300
40 project          = "example" ←
41 create_kms       = false
42

```

Figure 5 Changes in tfvars (infra directory)

And that is all for configuration, we are all set to deploy the tool and get cracking....but if there are any concerns, the table on the next page should sort it out.

Variable	Standard
----------	----------

namespace	Should be lowercase and unique e.g. xc3
env	Should be an environment where deployment will be done e.g. dev
project	The name of the project. It can be for any individual or group. It should be lowercase e.g. xc3
creator_email	Email address of the individual provisioning the resource. (Add personal email)
owner_email	Email address of the individual or team provisioning the resource. (Add personal email)
account_id	The account ID for your AWS account
domain_name	Domain name for the grafana dashboard (Optional, Not Recommended)
hosted_zone_id	Hosted Zone ID copied from Route 53 (Optional, Not Recommended)
ses_email_address	Email address of the individual provisioning the resource (Add personal email)
region	<pre> You, 1 second ago 2 authors (You and others) 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 = "Z053166920VP1STI0EK5X" 26 You, 1 second ago 2 authors (Wajahat Muahmmad and others) 27 private_subnet_cidr_block = { 28 "ap-southeast-2a" = "10.0.100.0/24" ← 29 } </pre>

Let Us Deploy!!!!

DEPLOTING THE TOOL

Now, let us integrate the XC3 infrastructure with your cloud AWS) infrastructure, so that our detective (XC3) can keep a private eye on all things cloud.

CREATING KEY PAIR FOR EC2 INSTANCE

The EC2 instance that will host XC3 will need a key pair, so let us give it one by parsing a .sh file and saving the key pair on the Linux subsystem, in the 'xc3/ infrastructure' directory

```
cd xc3/infrastructure/  
bash pre_req.sh  
chmod 400 "keypair.pem"
```

A new key pair with `should` now be created (replace keypair name with 'namespace-key.pem', like:

```
iamrrm-key.pem
```

APPLYING TERRAFORM MODULES (ACTUAL DEPLOYMENT/ INSTALLATION)

- Now, navigate to 'pre_requirement' directory and apply using

```
cd pre_requirement  
terraform init  
terraform plan -var-file=terraform.auto.tfvars  
terraform apply -var-file=terraform.auto.tfvars
```

One module should be successfully added post these commands.

- Now, navigate to 'infrastructure' directory and apply using

```
cd infrastructure  
terraform init  
terraform workspace new unique_name  
terraform plan -var-file=terraform.auto.tfvars  
terraform apply -var-file=terraform.auto.tfvars
```

The message describing **141 resources have been added** is all you need, **THE TOOL IS INSTALLED**

Apply complete! Resources: 141 added, 0 changed, 0 destroyed.

Outputs:

`xc3_url = "3.25.80.71"`

Figure 6 XC3 deployed

(To check if the resources have been deployed properly, you can navigate to your AWS account via the Management Console and to the **Lambda console**, where all **functions with your namespace** should be listed)

If installed properly, the Lambda function list should look like:

Functions (17)

Function name	Description	Package type	Runtime	Last modified
iamrm-resource-link-lambda		Zip	Python 3.9	2 hours ago
iamrm-project-spend-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-most-expensive-service-lambda		Zip	Python 3.9	2 hours ago
iamrm-project-spend-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-resource-link-lambda		Zip	Python 3.9	2 hours ago
iamrm-roles-list-lambda		Zip	Python 3.9	2 hours ago
iamrm-total-account-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-project-spend-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-most-expensive-service-lambda		Zip	Python 3.9	2 hours ago
iamrm-resource-link-lambda		Zip	Python 3.9	2 hours ago
iamrm-project-spend-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-most-expensive-service-lambda		Zip	Python 3.9	2 hours ago
iamrm-roles-list-lambda		Zip	Python 3.9	2 hours ago
iamrm-total-account-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-project-spend-cost-lambda		Zip	Python 3.9	2 hours ago
iamrm-most-expensive-service-lambda		Zip	Python 3.9	2 hours ago

Figure 7 XC3 deployed properly

And the Event Bridge rules should be visible in the Cloud Watch console as well:

Amazon EventBridge

Event bus: default

Rules (7)

Name	Status	Type	ARN	Description
iamrm-list-linked-account-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-list-linked-account-rule	Trigger the Lambda function once in a month
iamrm-most-expensive-service-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-most-expensive-service-rule	Trigger the Lambda function every week on Monday
iamrm-project-spend-cost-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-project-spend-cost-rule	Trigger the Lambda function every two weeks
iamrm-resource-list-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-resource-list-rule	Trigger the Lambda function every week on Monday
iamrm-roles-list-lambda-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-roles-list-lambda-rule	Trigger the Lambda function every week on Monday
iamrm-total-account-cost-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-total-account-cost-rule	Trigger the Lambda function every two weeks
iamrm-project-spend-cost-rule	Enabled	Scheduled Standard	arn:aws:events:ap-southeast-2:85172537:rule/iamrm-project-spend-cost-rule	Trigger the Lambda function every two weeks

Figure 8 Event Bridge Rule to Trigger 1st function

WHY THIS VERSION/ COMPONENT OF XC3

Let us see all the reasons why **this component/ version** of XC3 might be what you are looking for

FROM THE DEVELOPER'S NOTEBOOK

The **tool has been designed keeping in mind** that:

- **You are busy**, and must **not be burdened with** having to **trigger the Lambda** functions each time you want the updated costs for resources.
- You will have a **certain threshold** of **how often** you need **to check the updated costs** of your resources.

COMPONENT FEATURES

And the tool puts these assumptions in perspective to provide an **effective solution for resource cost management** by implementing the following:

- **Automating the** triggers the first Lambda function
- The first function whenever triggered (automated), will **set the whole workflow in motion automatically**, requiring no user intervention in the code or repository whatsoever.
- The **updated metrics** (like costs and resource names) will, due to the above automate workflow, will be **sent to Prometheus and then to Grafana automatically**, again requiring no intervention or config.
- The user manual gives detailed instruction for the user to **change the frequency of when and how many times the workflow runs**, giving flexibility and efficiency back to the user
- The infrastructure does not require any service other than Lambda and EC2 for basic functioning, so **no need for S3 or other storage solutions**
- There are multiple **bonus code snippets** in the repository, which are accessible for all that have enhanced and varied logic for solving the same problems, depending on the scenario and use case which the user can include (comment out the snippet of code).

For example, the logic for checking whether an S3 bucket is associated with an IAM Role has been included that checks the bucket policy instead of Role Policy Document.

FOR THE CLIENT:

- The **code style and structure** are **compatible with the existing XC3 repository structure**, basically following **least disturbance approach** for integration.
- The installation for the tools that have been developed as part of the individual contribution **can be installed using the same installation steps as provided for dev purposes**, with **not even a single extra step required** for basic installation of the updated tool.
- The **naming convention and resource definitions** in terraform are also **consistent with the existing XC3 repository**.
- The code is efficient and makes **use of resources, modules, and variables strictly when required** and **reuses** them whenever the need presents itself.

Now that you know why this version of XC3 is so special, **let us see what you need to do to get the tool running**.

HOW TO USE: TRIGGER AND OPERATE

JUST ACCESS THE DASBOARD WHENEVER YOU PLEASE!!!

(Note: **The other user manuals from group depict how to navigate the dashboard**)

Because the whole architecture is automated, there is no need for tuning the repository or logic anymore, the workflow is triggered automatically and the cost metrics are sent to the Grafana Dashboard, so just head straight to the dashboard and you will see the updated metrics for your AWS resources from the last time the workflow would have been triggered.

MANUAL TRIGGER

In case you want to manually trigger the workflow when an unforeseen event occurs and you need the updated costs before the next trigger is scheduled, just follow these steps:

- Navigate to Lambda Service via the AWS Management Console and look for the function named 'tour-namespace-roleslambda' which is the first Lambda function in the workflow.
- Click on the function and click on Test, which will prompt you to give it a test event name. Just choose any relevant name of your liking and invoke the function

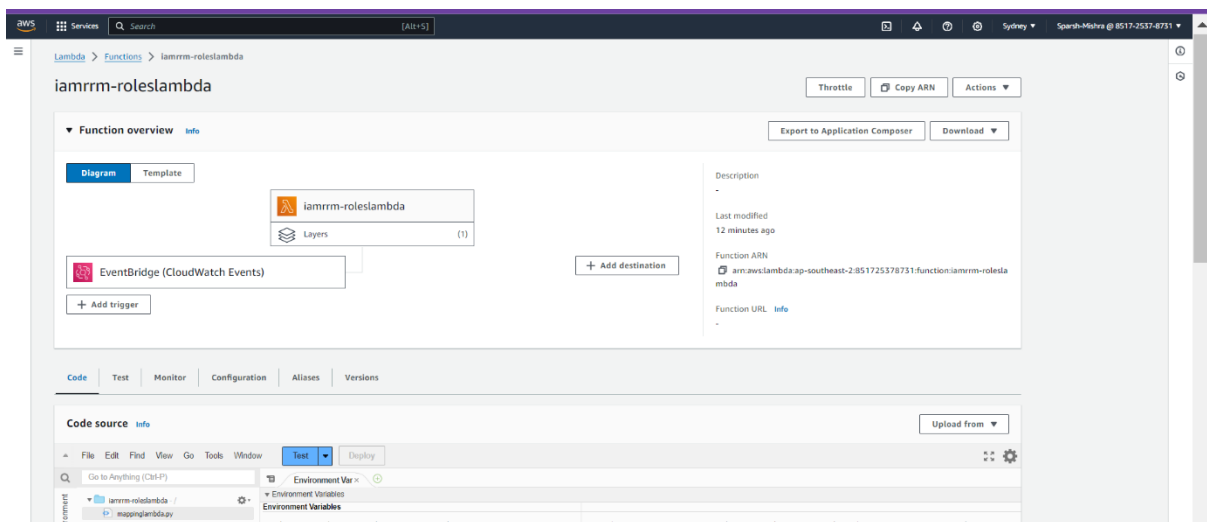


Figure 9 Manual Trigger

Since the next Lambda functions get triggered by this one, that is all you need to do. This trigger will lead to the workflow sending the latest CUR cost metrics to Prometheus, which will then be scraped by Grafana.

VERIFY PROPOER WORKING

To verify if the workflow has executed properly, there must be metrics showing in the Grafana dashboard that have been scraped from Prometheus, which can be verified in the following manner:

1. Access the Grafana dashboard using the following endpoint:
<http://<your-ec2-ip>:3000>, where <your-ec2-ip> is to be replaced with the IP of the instance where Prometheus and Grafana will be installed
2. Click on the hamburger option and click on Explore
3. Then in the Metric Explorer, type 'aim' and look for the metric named ;iam_role_resource_cost' as shown below:

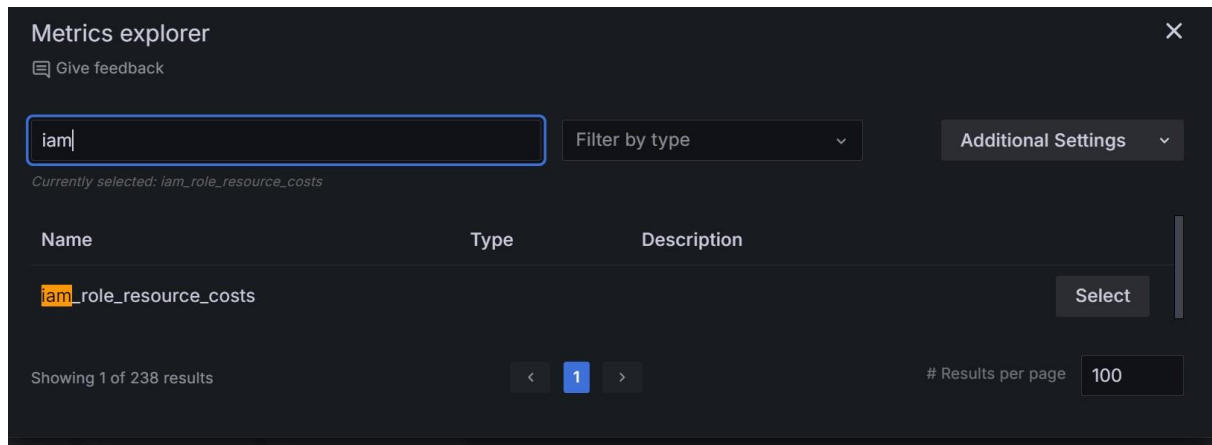


Figure 10 Looking for metric in Prometheus via Grafana

And that means that your tool (current component) is working just fine!!

You can explore the data/ metrics that have been sent from XC3 as well:

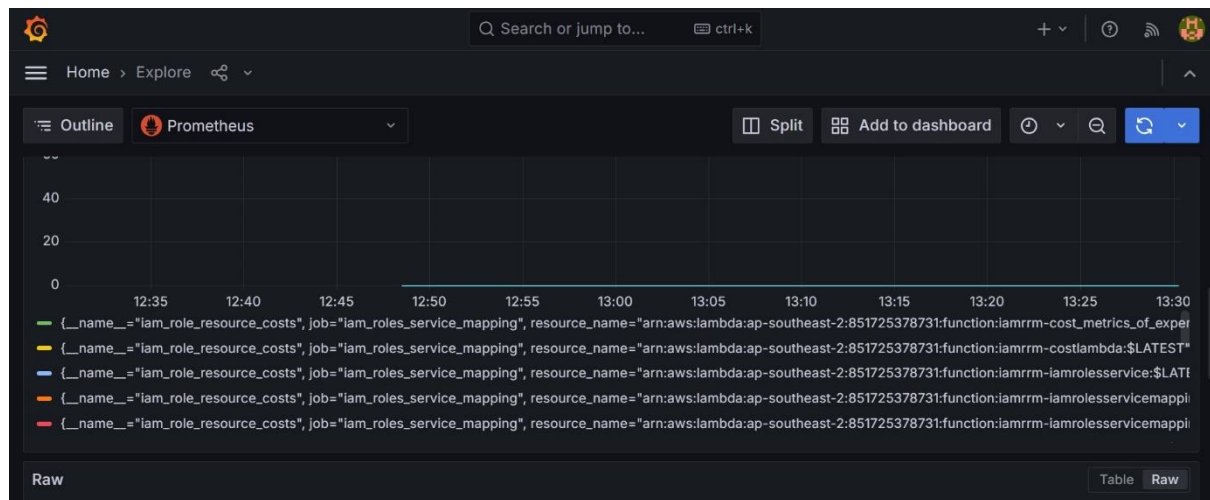
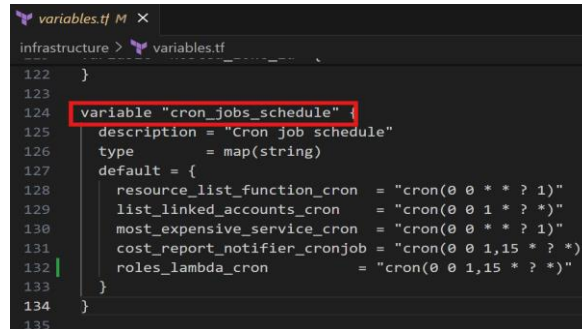


Figure 11 Exploring metrics

SETTING AUTOMATION PERIOD

The following steps will familiarise you with how you can change the periods and frequency of when the trigger for the first lambda function is automated for, thus automating the whole workflow. This will help you to save and utilize resources of XC3 only when you need then.

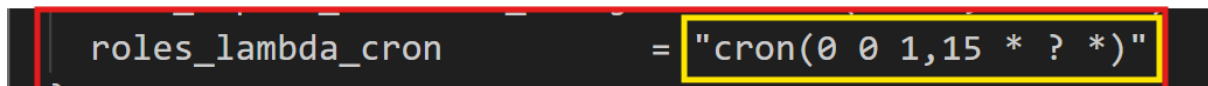
- Navigate to XC3 directory and then to xc3/infrastructure
- Open the 'variables.tf' file and look for the variable named 'cron_job_schedule'



```
122 }
123
124 variable "cron_jobs_schedule" {
125   description = "Cron job schedule"
126   type        = map(string)
127   default = {
128     resource_list_function_cron = "cron(0 0 * * ? 1)"
129     list_linked_accounts_cron   = "cron(0 0 1 * ? *)"
130     most_expensive_service_cron = "cron(0 0 * * ? 1)"
131     cost_report_notifier_cronjob = "cron(0 0 1,15 * ? *)"
132     roles_lambda_cron           = "cron(0 0 1,15 * ? *)"
133   }
134 }
```

Figure 12 Variable to Find

- In this variable, change the value assigned to 'roles_lambda_cron' to change the schedule following the below mentioned criterion:



```
roles_lambda_cron = "cron(0 0 1,15 * ? *)"
```

Here is an example to understand how to make changes:

The cron expression is in the format (Minutes Hours Day-of-month Month Day-of-week Year). For roles_lambda_cron, it may resemble cron(0 0 1,15 * ? *).

Here, 0 0 1,15 * ? * means:

At 00:00 (12:00 AM) on the 1st and 15th day of every month.

And here are some pre-defined values for your convenience:

Schedule	Cron Expression
Every 2 weeks	0 0 1,15 * *
Every 1 week	0 0 * * 0
Every 1 month	0 0 1 * *

Figure 13 Cron Job Schedules

DEBUGGING

On the little chance that the tool is not performing as expected, which can be indicated by:

- Grafana Dashboard saying 'No Data'.

This means that the dashboard is unable to fetch any data from Prometheus, indicating either the workflow has not been triggered or there are No IAM Roles to display data for

FIX

- a. Make sure there are IAM roles (at least one) that can be read and mapped by the workflow.
- b. Manually trigger the function using the steps in MANUAL TRIGGER.

- If you cannot access the dashboard and cannot see the functions in the Lambda Console or the EC2 instance, the tool has not deployed properly

FIX

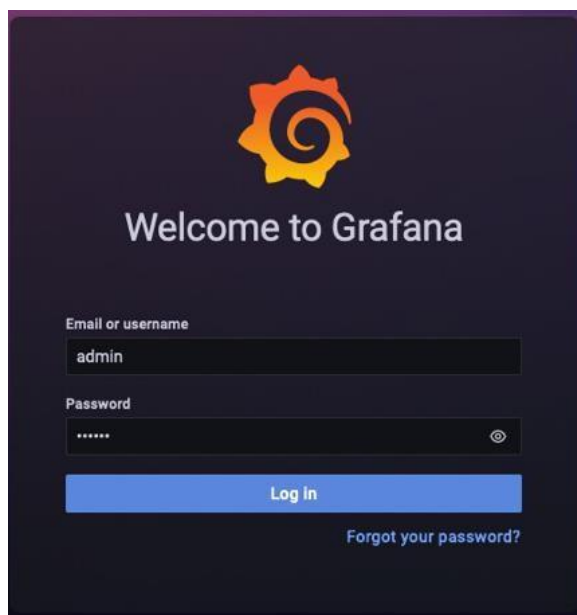
- a. Make sure you have stable internet connectivity.
- b. Destroy resources and deploy again.
- c. Do not turn off your PC or host while system is deleting or destroying.
- d. Refer to PRE-REQUIREMENTS and make your system meets them.

COMPONENT: DASHBOARD

HOW TO ACCESS AND NAVIGATE THE PANEL

ACCESS THE DASHBOARD

1. Go to the ec2 Public-ip:3000
2. Use the HTTP Protocol
3. Login to the dashboard with Grafana Default Credentials
4. Enter Username: admin
5. Enter Password: admin



Upon logging in, you will encounter various dashboards presenting different datasets. The "IAM Roles" dashboard primarily displays detailed information about IAM roles, their associated services, and their respective costs. Additionally, it highlights the top 5 resources for IAM role and service related to it. This comprehensive view enables users to understand the breakdown of costs and identify areas for optimization within their AWS environment.

IAM Role: This dropdown shows all IAM roles. Role

USING THE DROP DOWNS

1. Click the **IAM Role** dropdown menu.
2. Choose a specific role from the list.
3. The associated services will automatically populate in the **Service** dropdown.

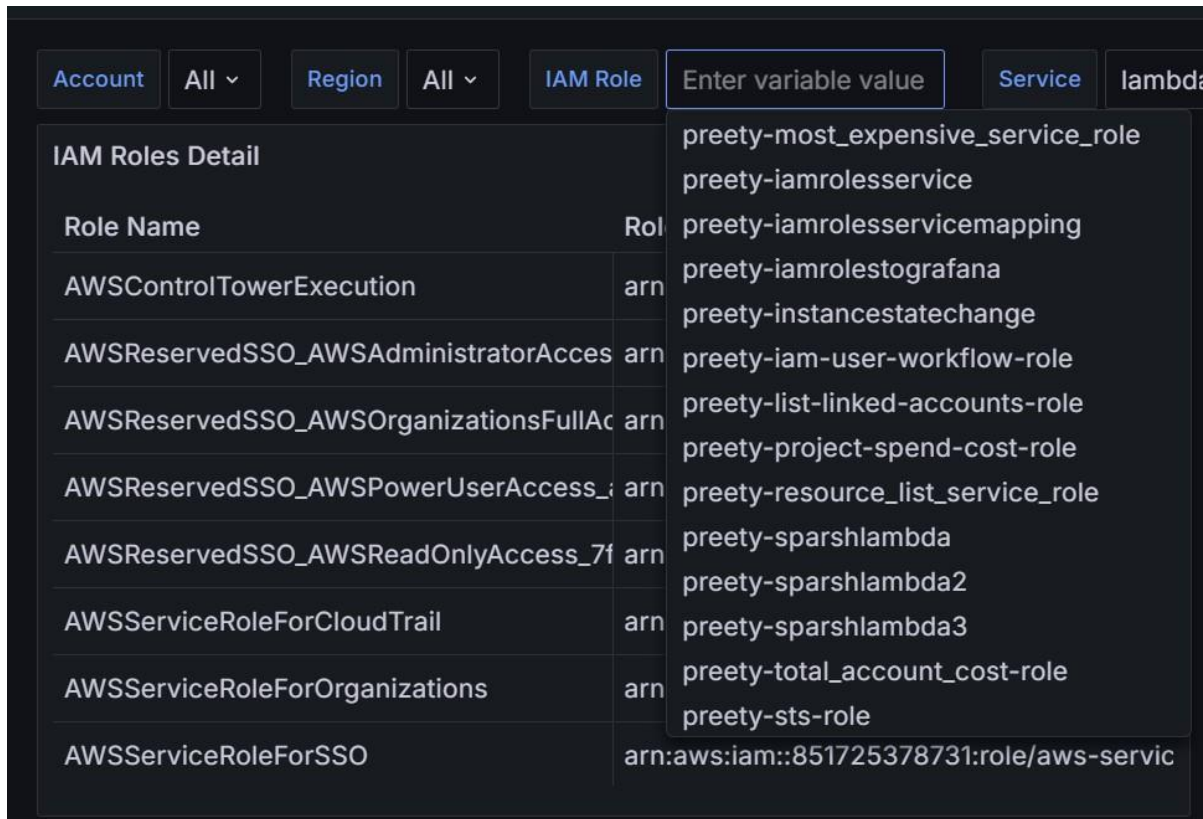


Fig: Drop down for IAM Role

Service: This dropdown shows the services associated with the role that is currently selected.

USING THE SERVICE DROP DOWN

1. After selecting an IAM role, pick a service from the Service dropdown.
2. You'll see the top 5 expensive resources related to that service.

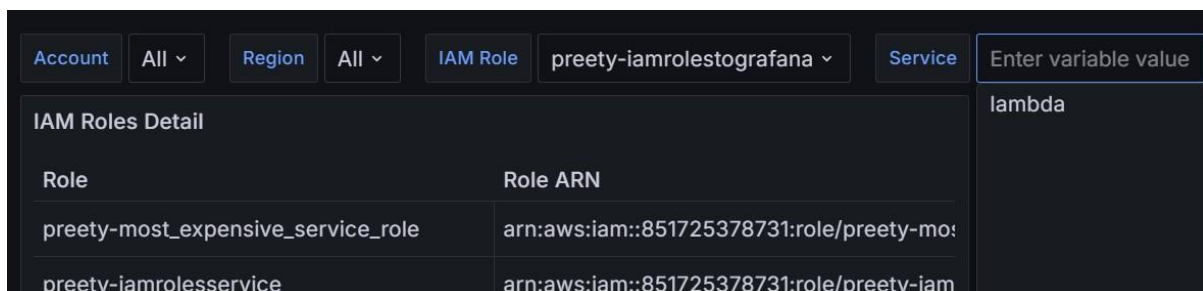


Fig: Dropdown for services

IAM ROLE DETAILS

The IAM roles detail is populated in the ‘IAM Roles Detail’ panel. The details are displayed in **table** format. These details will be shown automatically without any selection of **IAM Role** or **Service**.

Account	All ▾	Region	All ▾	IAM Role	preety-most_expensive_service_role ▾	Service	lambda ▾
IAM Roles Detail							
Role Name				Role ARN			
AWSControlTowerExecution				arn:aws:iam::851725378731:role/AWSControlTowerExecution			
AWSReservedSSO_AWSAdministratorAccess_12286c30830fafb1				arn:aws:iam::851725378731:role/aws-reserved/sso.amazonaws.com/ap-southeast-1/AWSI			
AWSReservedSSO_AWSOrganizationsFullAccess_a01dbc35bc9aac2f				arn:aws:iam::851725378731:role/aws-reserved/sso.amazonaws.com/ap-southeast-1/AWSI			
AWSReservedSSO_AWSPowerUserAccess_a84b45db0fb487cd				arn:aws:iam::851725378731:role/aws-reserved/sso.amazonaws.com/ap-southeast-1/AWSI			
AWSReservedSSO_AWSReadOnlyAccess_7f4e33ef26d15ee6				arn:aws:iam::851725378731:role/aws-reserved/sso.amazonaws.com/ap-southeast-1/AWSI			
AWSServiceRoleForCloudTrail				arn:aws:iam::851725378731:role/aws-service-role/cloudtrail.amazonaws.com/AWSService			
AWSServiceRoleForOrganizations				arn:aws:iam::851725378731:role/aws-service-role/organizations.amazonaws.com/AWSSer			
AWSServiceRoleForSSO				arn:aws:iam::851725378731:role/aws-service-role/sso.amazonaws.com/AWSServiceRoleF			
AWSServiceRoleForSupport				arn:aws:iam::851725378731:role/aws-service-role/support.amazonaws.com/AWSServiceR			

Fig: IAM Roles Detail panel

SERVICE COST PANEL

Once you have selected a **Role**, the details of services and their associated costs are in the ‘**Services and Cost**’ panel. The data is conveniently displayed in a **bar chart** format, allowing you to quickly analyze and understand the cost distribution.



Fig: Services and Cost panel representing services of selected IAM Role

Resource Detail	
Resource Name	Cost
dummy-bucket0	\$21.7
dummy-bucket1	\$41.3
dummy-bucket2	\$12.7
dummy-bucket3	\$40.8
dummy-bucket4	\$15.2
dummy-bucket5	\$32.3

Upon selecting a role and its corresponding service, two informative panels will be presented. These panels provide essential details and guidance related to your chosen role and service.

Top 5 expensive resources for (Service): This panel exhibits the top five expensive resources for the selected service through a bar chart visualization.

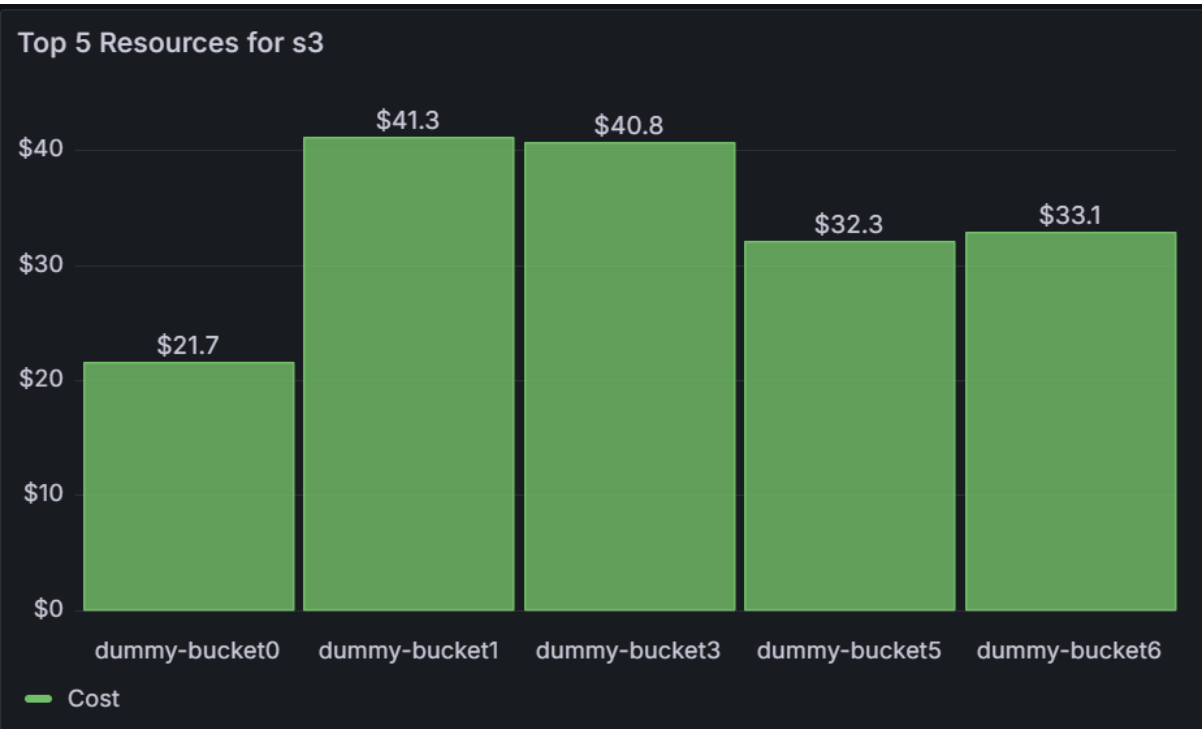


Fig: Top 5 expensive Resources for the selected service

Resource Detail panel: This panel shows details of all resources related to the service selected for the particular IAM Role. It shows the Resource Name and Cost of the resource visualized in table format.

ADDITIONAL CHANGES MADE

In this section, the components that have been discussed above have been elaborated upon from the perspective of the developer (myself). This is also to depict the changes in the original XC3 repository I have made.

You can check out my individual branch at:

<https://github.com/104202994/xc3/tree/sparsh>

LAMBDA 1 – ROLES LAMBDA

This is the overview of the first Lambda function in the workflow when the automation CloudWatch mechanism triggers it:

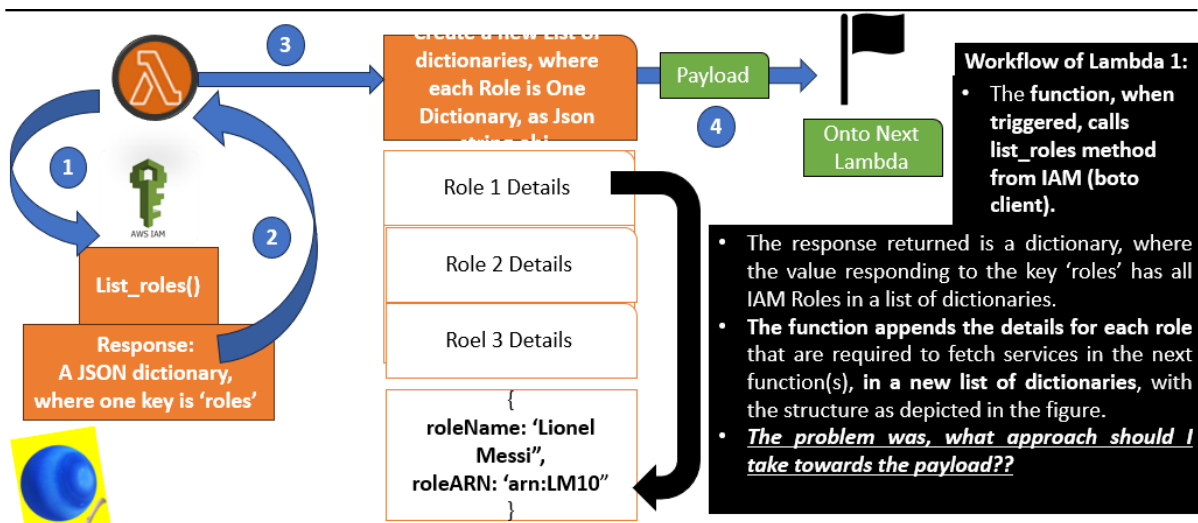


Figure 14 LAMBDA 1

The first function gets the list of IAM Roles and passes it as payload to the next function. The code snippets have been attached below:

```
def get_iam_role_details():
    iam_client = boto3.client('iam')

    # List all IAM roles in the current AWS account
    response = iam_client.list_roles()

    # Extract details for each IAM role
    iam_roles = []
    for role in response['Roles']:
        role_name = role['RoleName']
        role_arn = role['Arn']
        role_creation_date = role['CreateDate'].strftime('%Y-%m-%d %H:%M:%S')

        iam_roles.append({
            'RoleName': role_name,
            'RoleArn': role_arn,
            'CreationDate': role_creation_date,
            'Description': role_description,
            'Path': role_path,
            # Add more fields as needed
        })

    return iam_roles
```

```
# Prepare the payload to send to lambda2
payload = {
    "iam_roles": iam_roles
}

# Convert the payload to JSON
json_payload = json.dumps(payload)

# Invoke lambda2
lambda_client = boto3.client('lambda')
response = lambda_client.invoke(
    FunctionName= os.environ["func_name_mapping_lambda"],
    InvocationType='Event', # Synchronous invocation
    Payload=json_payload
)
```

Figure 15 LAMBDA 1 CODE

LAMBDA 2 –COST LAMBDA

This is the second function in the workflow that maps services and resources to the role dictionary that was first revived as payload from the first Lambda function as explained below:

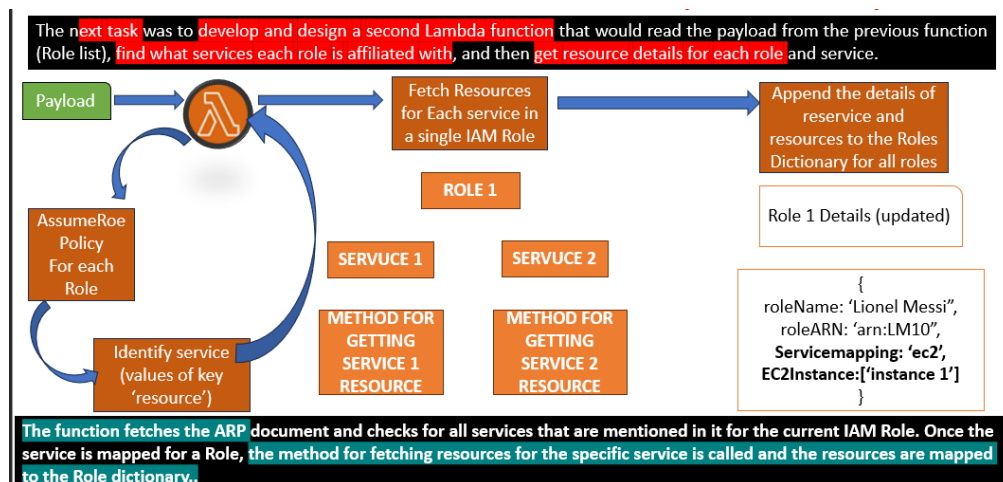


Figure 16 LAMBDA 2

This function does the following:

- Fetch AssumeRolePolicy Document for each role in the list of dictionaries passed as payload from the first function.
- Search for affiliated services in the Document and map the service to the Role dictionary in the Service Mapping key, the value for which will be the service name like 'ec2', 'lambda' etc, and No Services if there are no services attached to the Role.
- Then, for each service (except RDS, as RDS developed by team mate, not myself), a separate method is called that fetches the resource(s) for that service, for the current role, and then this service is mapped to the role dictionary.
- This updated list of Roles with the services and resources mapped is then passed as payload to the third Lambda functions.

(I have incorporated EC2, S3, LAMBDA AND DYNAMO into the mechanism, branch ULRL in the end)

The following code snippets depict the above workflow for Lambda service and functions:

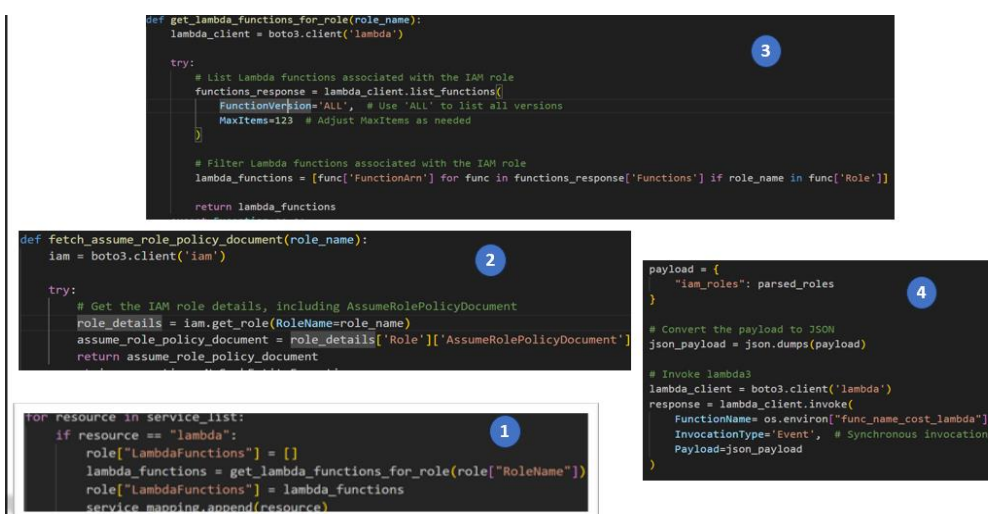


Figure 17 LAMBDA 2 CODE

The resource mapping methods that I had developed for **Lambda and EC2** are straight forward:

- Once the service is identified to be EC2 or Lambda for a role, the specific service is called for all resources it has (functions and instances respectively) and the resources are then checked for the role association, and if found, they are mapped to the role dictionary. (For EC2, the instance profiles are fetched to then be matched against the instance)

The methods for S3 and dynamo are more complex, where:

Once S3 or Dynamo is identified to be associated with an IAM Role, all associated policies for the Role are etched, and the policy document that are under the 'Allow' effect are then checked for 's3' or 'dynamo' key, after which the tables can be fetched by navigating to the Resource key in the policy document for the respective service associated with the role.

AUTOMATION MECHANISM

The automation mechanism that I have incorporated in the project can be understood as a separate component. This component in the context of the project can be understood in the following manner:

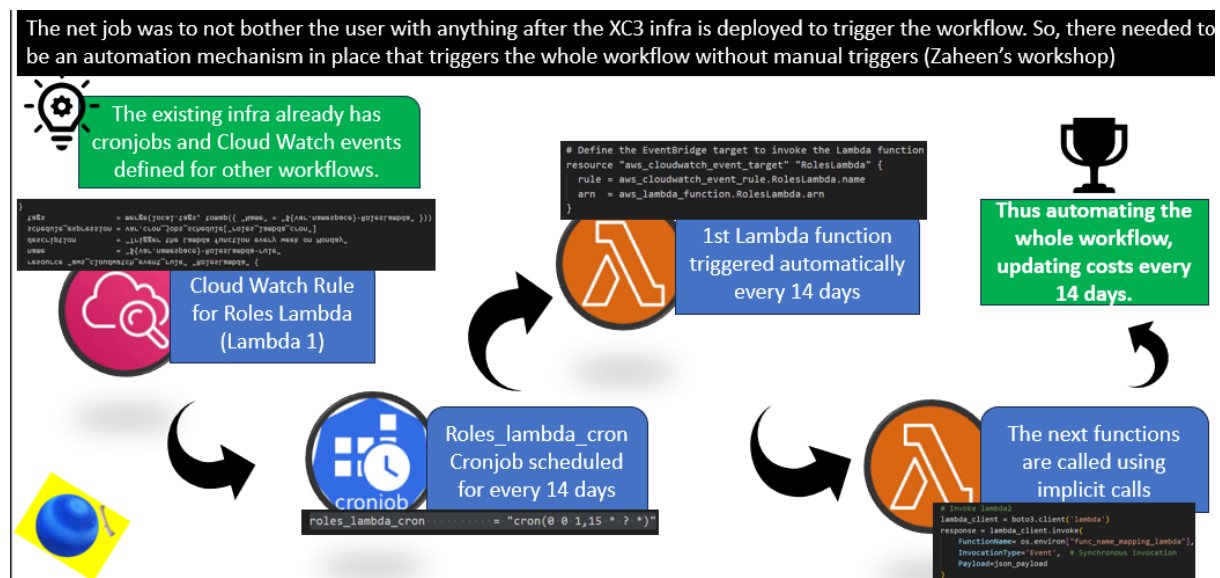


Figure 18 Automation workflow

The automation workflow triggers the first function in accordance with the cron job schedule, which in turn triggers the second, and the second triggers the third, achieving complete workflow automation.

The snippets of code above are the updates that I have made in the iam-role.tf file in the 'infrastructure / modules / serverless' directory, along with the cron job schedule variable added to the 'infrastructure / variables.tf' file.

```

resource "aws_lambda_permission" "allow_bucket_for_irtg" {
  source_arn = var.s3_xc3_bucket.arn
}

resource "aws_cloudwatch_event_rule" "RolesLambda" {
  name           = "${var.namespace}-RolesLambda-rule"
  description    = "Trigger the Lambda function every week on Monday"
  schedule_expression = var.cron_jobs_schedule["roles_lambda_cron"]
  tags           = merge(local.tags, tomap({ "Name" = "${var.namespace}-RolesLambda" }))
}

# Define the EventBridge target to invoke the Lambda function
resource "aws_cloudwatch_event_target" "RolesLambda" {
  rule = aws_cloudwatch_event_rule.RolesLambda.name
  arn  = aws_lambda_function.RolesLambda.arn
}

resource "aws_lambda_permission" "RolesLambda" {
  statement_id = "AllowExecutionFromEventBridge"
  action       = "lambda:InvokeFunction"
  function_name = aws_lambda_function.RolesLambda.function_name
  principal     = "events.amazonaws.com"
  source_arn    = aws_cloudwatch_event_rule.RolesLambda.arn
}

```

Figure 19 Changes in iam-role.tf file

TERRAFORM : RESOURCE DEFINITIONS AND CONFIG

The following is a description of terraform configurations that I added to the repository for the Lambda functions that I have incorporated into the tool and have discussed above, along with a third function that someone else developed as part of the team effort.

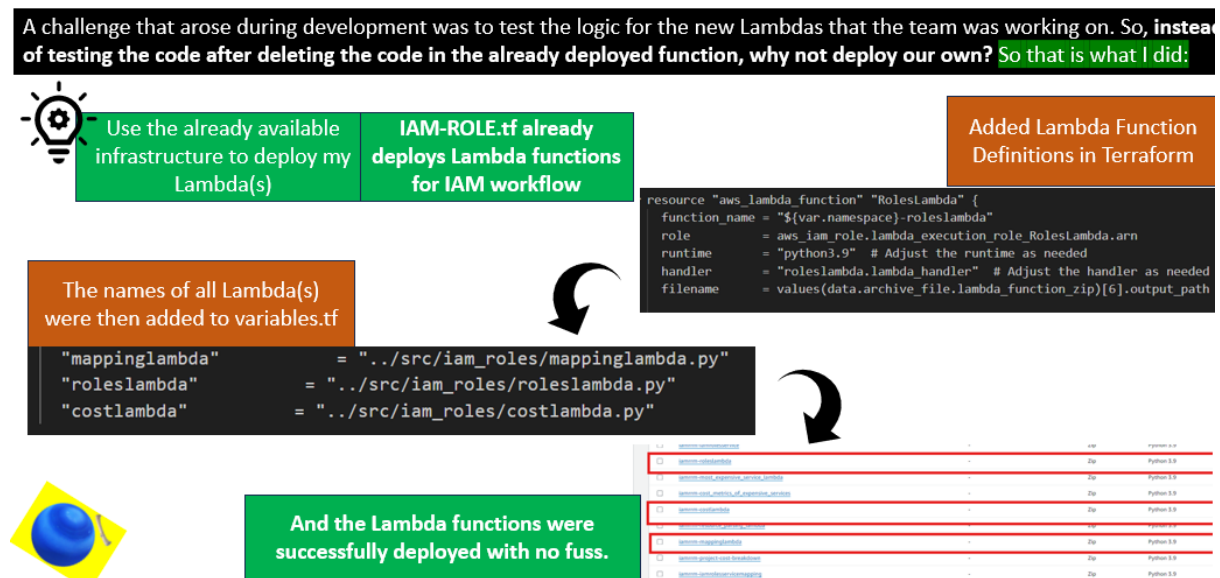


Figure 20 Terraform Configurations

The configurations that I added for the three functions included the following for each function:

- The resource/ function definition that defines the lambda function

```

resource "aws_lambda_function" "CostLambda" {
  function_name = "${var.namespace}-costlambda"
  role          = aws_iam_role.lambda_execution_role_CostLambda.arn
  runtime       = "python3.9" # Adjust the runtime as needed
  handler       = "costlambda.lambda_handler" # Adjust the handler as needed
  filename      = values(data.archive_file.lambda_function_zip)[0].output_path

  environment {
    variables = {
      # Add any environment variables specific to CostLambda function
      prometheus_ip = "${var.prometheus_ip}:9091"
      region_names_path = "${var.namespace}/region_names"
      bucket            = "${var.namespace}-metadata-storage"
    }
  }

  memory_size = var.memory_size
  timeout      = var.timeout
  vpc_config {
    subnet_ids      = [var.subnet_id[0]]
    security_group_ids = [var.security_group_id]
  }

  layers = [var.prometheus_layer]

  tags = merge(local.tags, tomap({ "Name" = "${var.namespace}-cost_lambda" }))
}

```

Figure 21 Function definition

- The role for the lambda function

```

# Creating IAM Role for CostLambda function
resource "aws_iam_role" "lambda_execution_role_CostLambda" {
  name = "${var.namespace}-costlambda"
  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Action = "sts:AssumeRole",
        Effect = "Allow",
        Sid    = "costlambdareole",
        Principal = {
          Service = "lambda.amazonaws.com"
        }
      }
    ]
  })
  managed_policy_arns = [
    "arn:aws:iam::aws:policy/AmazonS3FullAccess",
    "arn:aws:iam::aws:policy/IAMFullAccess",
    "arn:aws:iam::aws:policy/CloudWatchReadOnlyAccess",
  ]
  tags = merge(local.tags, tomap({ "Name" = "${var.namespace}-IAM-Role-CostLambda" }))
}

```

Figure 22 Role for Lambda

- The role policy for the role of the lambda function

```

resource "aws_iam_role_policy" "CostLambda" {
  name = "${var.namespace}-costlambda-lambda-inline-policy"
  role = aws_iam_role.lambda_execution_role_CostLambda.id

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Action = [
          "s3:*", # Full S3 access
          "iam:*", # Full IAM access
          "ssm:Get*",
          "ssm:List*",
          "ce:GetCostAndUsageWithResources", # CloudWatch Cost Explorer
          "ec2:*",
          "lambda:InvokeFunction", # Allow invoking Lambda functions
          "lambda:*"
        ],
        Resource = ["*"]
      }
    ]
  })
}

```

Figure 23 Role policy for Lambda role

COST CALCULATION RESOURCES (EC2, S3, LAMBDA, DYNAMO)

I had also developed the methods for getting costs for all resources from 4 services (all except RDS in the repository) which was then incorporated into the third Lambda function by a group mate.

The idea is simple and this is the approach my cost calculation methods take:

- The resource identifier for the resource (ARN or name depending on service) is passed as a parameter to the function along with the start and end dates (discussed above)
- The method calls the navigates to the CUR using the Cost Explorer client and fetches the unblended cost by filtering the report according to the dates and looking for the resource.

The following snippet depicts how I did it for Lambda (cost calculation logic for resources of all services can be found the branch in the src/iam-role/ vostlambda.py' file):

```
def get_lambda_function_cost(lambda_function_arn, start_date, end_date):
    cost_client = boto3.client('ce')

    try:
        cost_response = cost_client.get_cost_and_usage(
            TimePeriod={
                'Start': start_date,
                'End': end_date
            },
            Granularity='DAILY',
            Metrics=['UnblendedCost'],
            Filter={
                "Dimensions": {
                    "Key": "RESOURCE_ID",
                    "Values": [lambda_function_arn]
                }
            }
        )

        costs = cost_response['ResultsByTime'][0]['Total']['UnblendedCost']['Amount']
        costs = 0 if costs is None else float(costs)

        return costs
    except Exception as e:
        print(f"Error fetching cost for Lambda function '{lambda_function_arn}': {str(e)}")
        return 0
```

Figure 24 Cost Calculation

PAYLOAD MECHANISM

I had made an MVP that was achieving all required requirements up until the point of cost calculation(when I was in a group of two), but the mechanism was then using an S3 bucket to store the roles in a JSON file, from where the second Lambda function was fetching this file and then parsing the roles further for getting resources and then services.

One of the longest running and bothersome issue that I came across was to be able to pick how multiple lambda functions in my workflow will communicate: **dynamically or statically?** Let me explain

- This approach has each Lambda function send a JSON file to an S3 bucket with the updated details for Roles.

- The next function then navigates to the same S3 bucket and directory (key) and fetches the same file, reads it and then proceeds.

Why put an S3 bucket in the middle? To manage and debug payload issues and identify logic issues more conveniently



- This approach sends a JSON payload straight away to the next Lambda function when invoking it from the first Lambda (implicit call)

Why direct payloads? The approach is easier, direct and the scope is restricted to just Lambda service (no s3 or dynamo)

The winner is.... OPTION 2

Figure 25 aS3 vs. Payloads

But since this might lead to incurred costs when the number of roles grows increasing the file size, and the time it will take to upload this file in the S3 bucket, I changes the mechanism to not require an S3 bucket and just be able to work with payloads, and I incorporated this into the third functions as well, which was developed by someone else.

COST BREAKDOWN FOR THE PROPOSED ARCHITECTURE

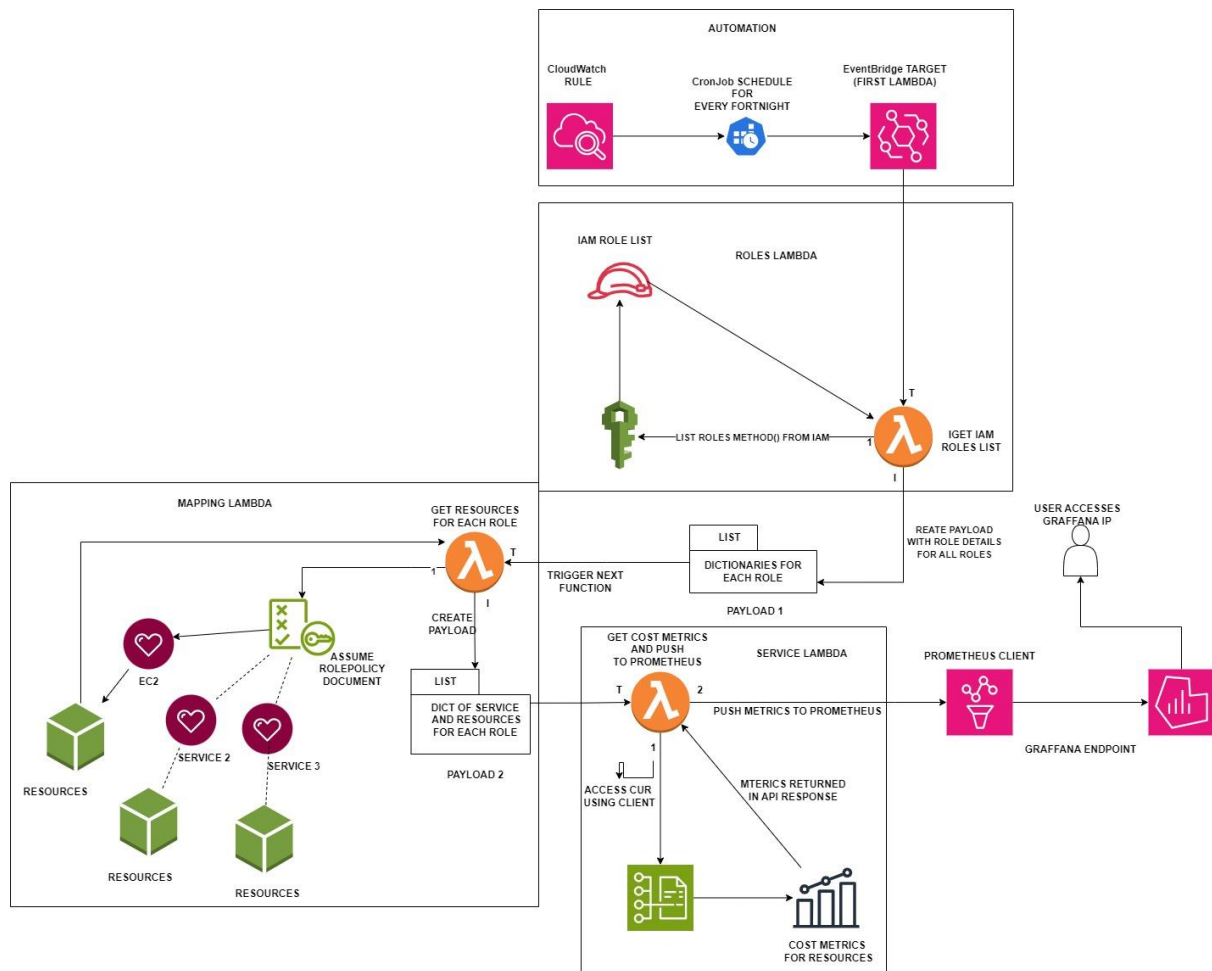


Figure 26 Final Proposed Architecture

Cost and Usage Report (Estimated)

Assumptions:

20 IAM roles

Lambda functions invoked every 14 days

Free tier applies to most services

Costs:

Lambda Invocations:

First Lambda (3 seconds @ 128MB): ~\$0.000024 per execution
(<https://aws.amazon.com/lambda/pricing/>)

Second Lambda (20 seconds @ 128MB): ~\$0.00016 per execution

Third Lambda (20 seconds @ 128MB): ~\$0.00016 per execution

Total Lambda cost per invocation: ~\$0.000344 (all functions)

Note: This is a very small cost, but it can add up over time depending on the frequency of invocations.

S3 Storage:

Cost depends on data size: Check S3 pricing for specifics (<https://aws.amazon.com/s3/pricing/>)

Example: 1 GB of data stored in S3 Standard Storage for a month might cost around \$0.023. Actual cost will vary based on your data size and storage class.

S3 Get/Put Objects:

Retrieving and updating the report incurs minimal costs based on requests and data transferred. Difficult to estimate without specific data size. Expect very low costs.

CloudWatch Events:

Free tier likely covers this usage. Check the latest pricing details for CloudWatch Events if exceeding the free tier is a concern (<https://aws.amazon.com/cloudwatch/pricing/>).

Data Transfer:

Lambda-to-Prometheus Pushgateway:

Free if Pushgateway is in the same VPC.

If outside (e.g., public subnet), inter-AZ or internet data transfer charges apply. Costs depend on data size and transfer zone. Expect minimal costs for small data transfers within the same region.

Pushgateway-to-Prometheus: Usually free as it's likely within the same VPC.

Third Lambda to EC2 Instance:

Cost depends on the size of the matrix data pushed. Check EC2 data transfer pricing (<https://aws.amazon.com/ec2/pricing/>). Difficult to estimate without data size.

EC2 Instance:

Cost depends on instance type and usage: Choose a right-sized instance (CPU, memory, disk) for your needs. Terminate or stop unused instances. Refer to EC2 pricing for details (<https://aws.amazon.com/ec2/pricing/>).

Example: An t2.micro instance running on-demand in US East (N. Virginia) could cost around \$0.0126 per hour (<https://aws.amazon.com/ec2/pricing/>). Actual cost will vary based on your chosen instance type and usage pattern.

Remember: These are just estimates. The actual costs will depend on your specific usage patterns and data sizes.

REFERENCES

Irfan, M. (n.d.). *Enhancing IAM Role-Based Cost Breakdown Feature in XC3*. Retrieved from GitHub XC3: <https://github.com/XgridInc/xc3/issues/143>

Mathew, A. (2021). *Exploring the cost and performance benefits of AWS step functions using a data processing pipeline*. ACN Digital Library.

Resource-level data at daily granularity. (n.d.). Retrieved from AWS Documentation: <https://docs.aws.amazon.com/cost-management/latest/userguide/ce-resource-daily.html>

Variables. (n.d.). Retrieved from Grafana Documentation: <https://grafana.com/docs/grafana/latest/dashboards/variables/>

Working with binary payloads. (n.d.). Retrieved from AWS Documentation: <https://docs.aws.amazon.com/iot/latest/developerguide/binary-payloads.html>