CLOUDSPIKES

Multi-cloud solutions

AWS Cloud Infra Training Outline - 13 Tasks

CLOUDSPIKES

Multi-cloud solutions

+1-647-376-7753 | support@cloudspikes.ca | www.cloudspikes.ca

AWS Cloud Infra Tasks - Index

#1 Task: Have a look at the Cloud expenses from Cost Explorer regularly. Also, enable MFA for the AWS Root User.

Cost Analysis Tools on AWS

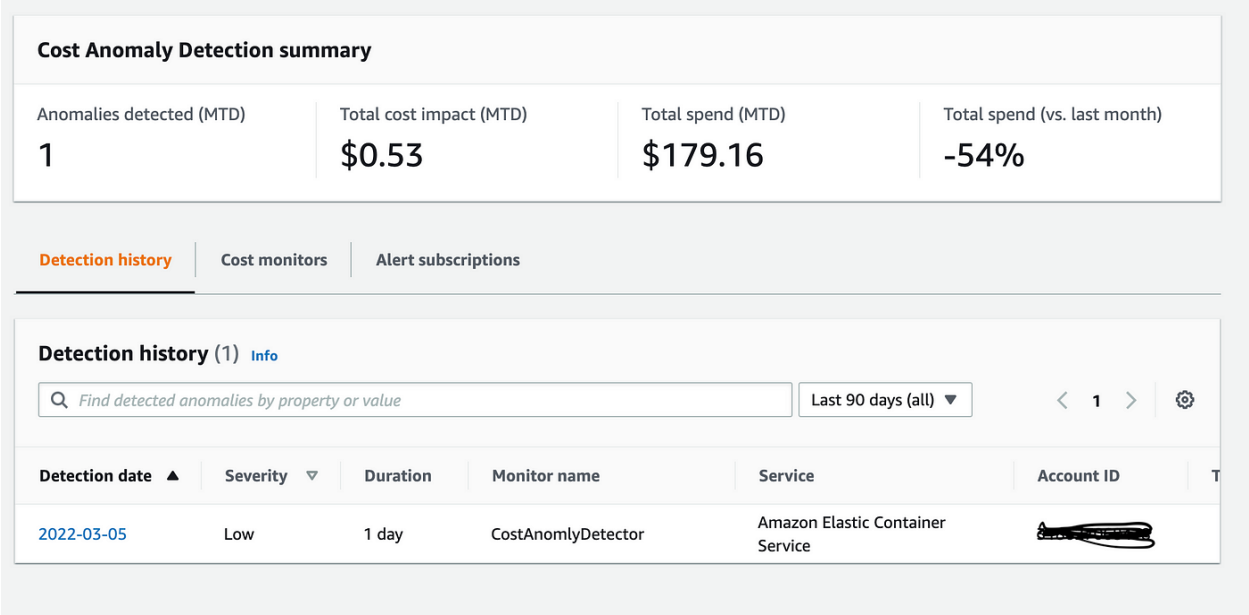
We will explore some of the AWS tools which can be used to understand the cost and also create some notifications-based alerts to tame the usage of the AWS services.

- Cost Anomaly Detection
- AWS Cost Explorer
- AWS Budget
- AWS Cost and Usage Dashboards

Cost Anomaly Detection

AWS Cost Anomaly Detection is a monitoring feature that utilizes advanced machine learning techniques that identify anomalous and suspicious spend behaviors as early as possible so we can avoid costly surprises. Based on the selected spend segments, Cost Anomaly Detection automatically determines patterns each day by

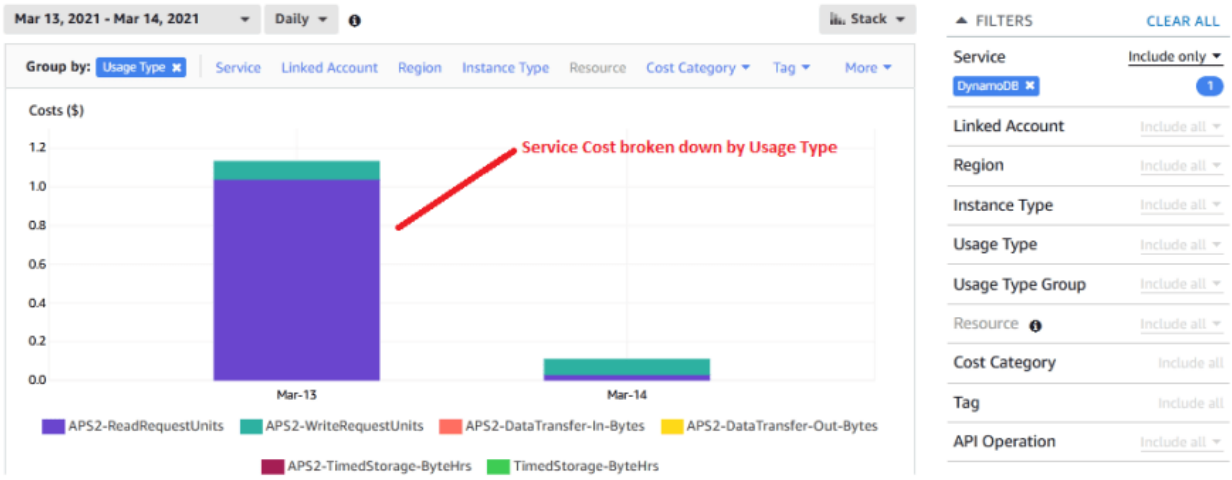
adjusting for organic growth and seasonal trends. It triggers an alert when spend seems abnormal.



AWS Cost Explorer

The Cost Explorer provides us with granular insight into the AWS service usage and costs accrued. It gives us a detailed breakdown of all services in the table view & visually. We can get AWS costs and usage over a daily or monthly granularity. It gives us the capability to create reports.

With granular insights, we can identify services and usage types that are costing more. It also helps us visualize cost trends over time. We can look for any sudden spike in usage.



AWS Budget

Sitting alongside each other within the AWS Cost Management group, AWS Budgets and AWS Cost Explorer are complementary services from Amazon Web Services (AWS). Using them together, we can analyze our cost and usage patterns and use that analysis to implement effective governance controls and cost optimizations. AWS Budgets lets us track service usage, utilization, and coverage for Reserved Instances and Savings Plans.

Creating budgets

There is an AWS setup wizard that starts when we create a budget through the console. Within the wizard, there are “Info” links at the top of each page, which lead to more detailed instructions and AWS documentation.

Budget alerts

Alerts are attached to a budget and can be created when creating or editing that budget. They consist of a threshold and a notification. The threshold contains a trigger, which determines whether the alert fires when today’s actual usage or the forecast usage for the budget period crosses the threshold.
We can also configure alerts to trigger automated actions when they fire. An action can be one of:

1. Attach an IAM policy to a user, group, or role.

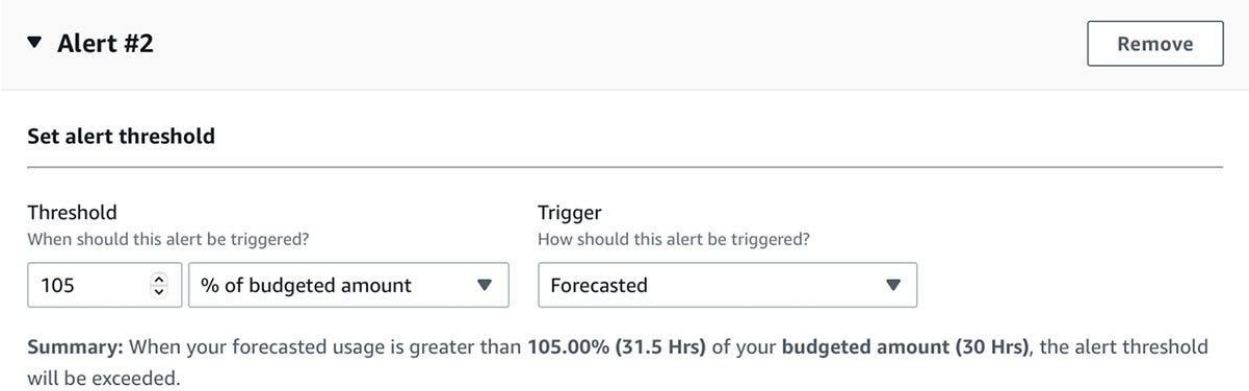
2. Attach a Service Control Policy (SCP) to an Organizational Unit or the organization's root.
3. Stop specific EC2 or RDS instances.

Forecasting in AWS Budgets

Forecasting gives us an additional option for how we set up our alerts. Using actual spending, we might set up an alert to notify us at 80% of our monthly budget, say. Using forecasting, we might want an alert based on projected overrun, say 105%. We can use both types of alerts with the same budget — a budget will support up to 5 alerts.

Limitations :

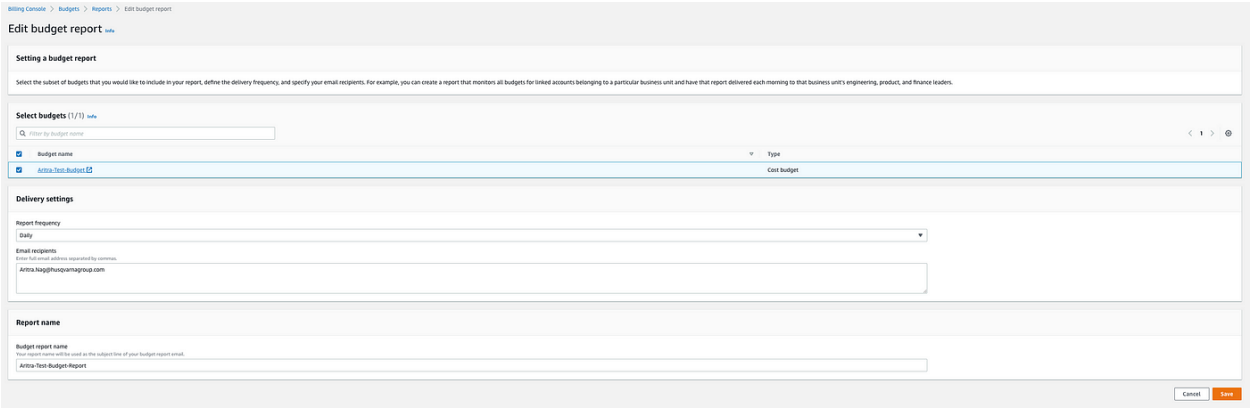
1. Forecast alerts won't fire at all unless we already have enough (roughly 5 weeks) usage data.
2. As with any forecasting, although there will be some intelligence in the underlying algorithm, it's only based on ourprevious usage patterns and may be inaccurate.



Budget Reports

Budget reports are sent by email on a regular cadence.
AWS keeps these reports simple (the only complexity is in the budgets themselves) — for a single report we can customize:

1. Which budgets are included (we can include more than one)?
2. How often the report should be sent — daily, weekly, or monthly.
3. Who should receive the report — a list of email addresses.



We will look into both services in more detail and see how they differ and how they can be used together.

Dimension	AWS Budgets	AWS Cost Explorer
Main use case	Governance controls	Cost analysis
Ease of use	• Simple user interface • Guided setup for reports and alerts	• Chart interface, with filter options • Multiple built-in charts that can be adapted
Useful features	• Regular report delivery • Granular filters • Alerts • Automated responses	• Data visualization • Granular filters • Cost-saving recommendations • Sharing reports • Hourly granularity
Customization	• Filtering • Email recipients • Automated responses	• Filtering and grouping

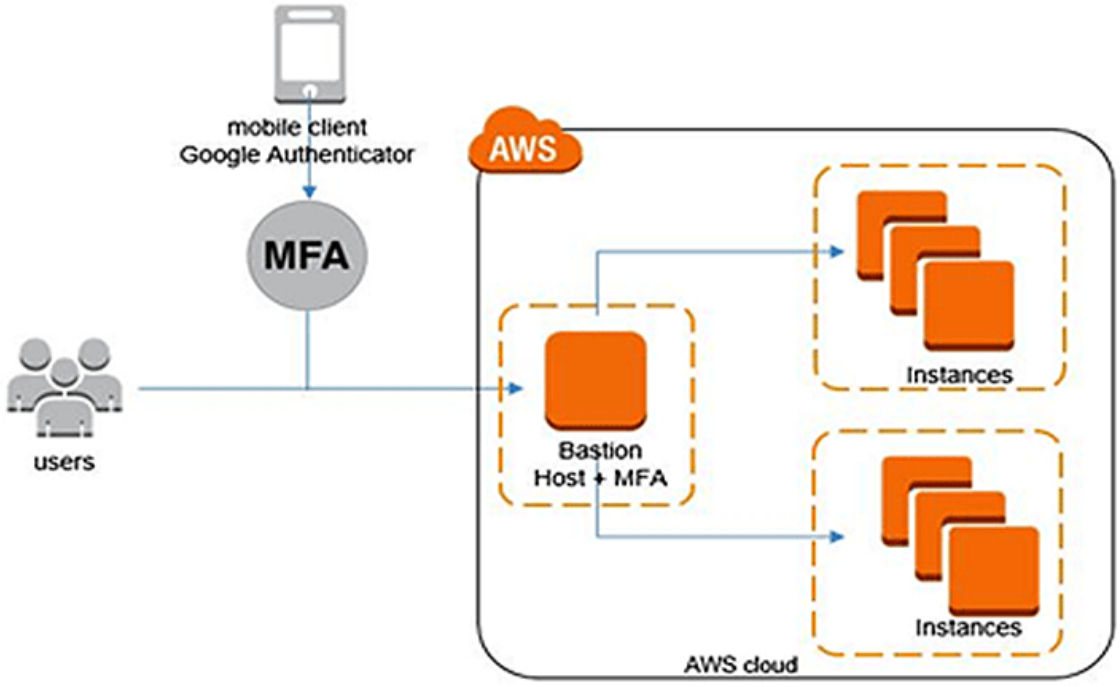
Enable MFA for the AWS Root User.

Overview of AWS MFA

AWS Multi-Factor Authentication (MFA) is a simple best practice that adds an extra layer of protection on top of your username and password. With MFA enabled, when a user signs in to an AWS Management Console, they will be prompted for their user name and password (the first factor is what they know), as well as for an authentication code from their AWS MFA device (the second factor is what they have). Taken together, these multiple factors provide increased security for your AWS account settings and resources.

Why AWS MFA is Required

- Users have access to your account and can possibly change configurations and delete resources in your AWS account, so to overcome this it is required
- If you want to protect your root accounts and IAM user.
- Even if the password is stolen or hacked, the account is not compromised.
- When you enable this authentication for the root user, it affects only the root user credentials. IAM users in the account are distinct identities with their own credentials, and each identity has its own MFA configuration.



MFA Device Options In AWS

The following are the MFA device options in AWS:

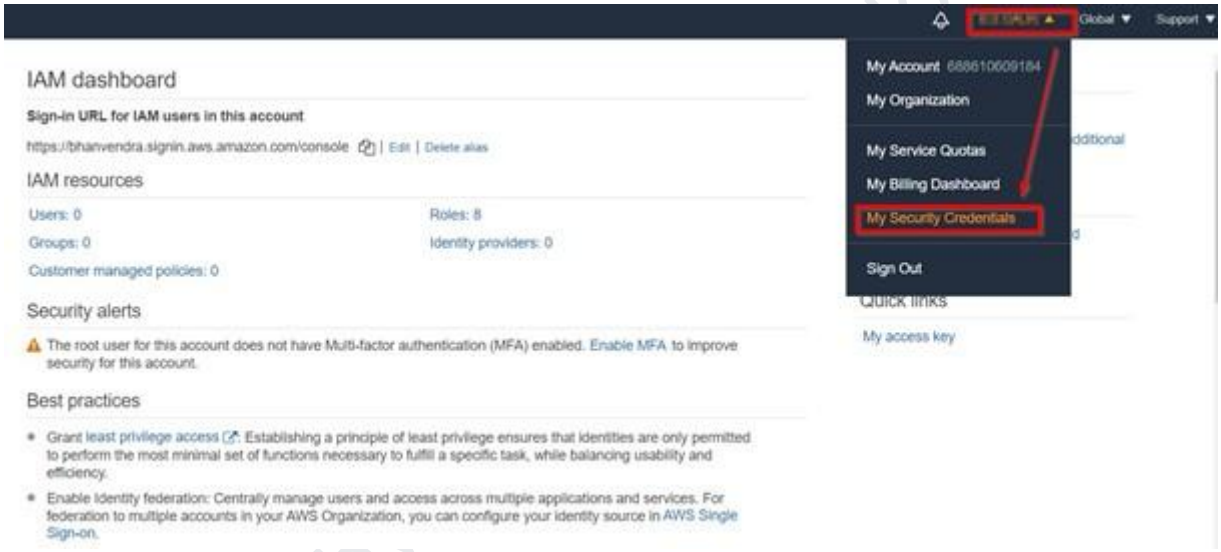
- Virtual MFA Device: Support for multiple tokens on a single device e.g Google Authenticator (Phone Only) Authy(Multi-Device)

- Universal 2nd Factor (U2F) Security Key: Supports multiple root and IAM users using a single security key. e.g Yubikey by Yubico (Third Party)
- Hardware Key Fob MFA Device: Provided by Gemalto (Third Party)
- Hardware Key Fob MFA Device AWS GovCloud (US): Provided by SurePassID (Third Party)

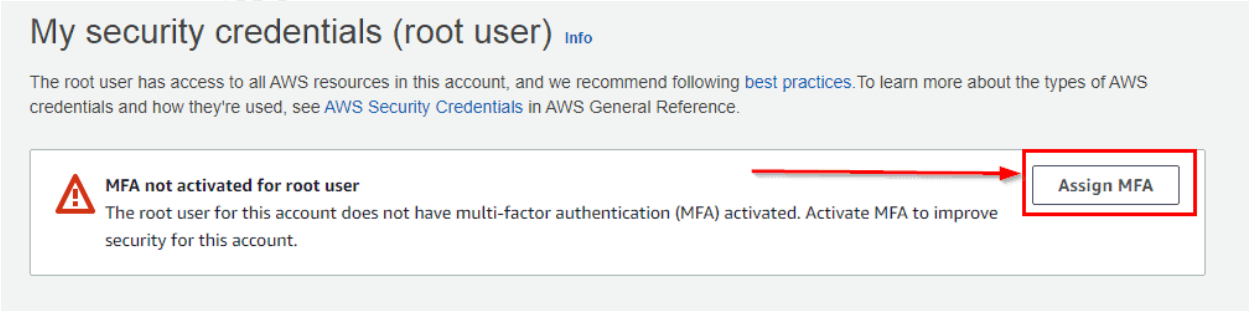
Enabling MFA On Root Account

1) Log in to your AWS account by clicking here

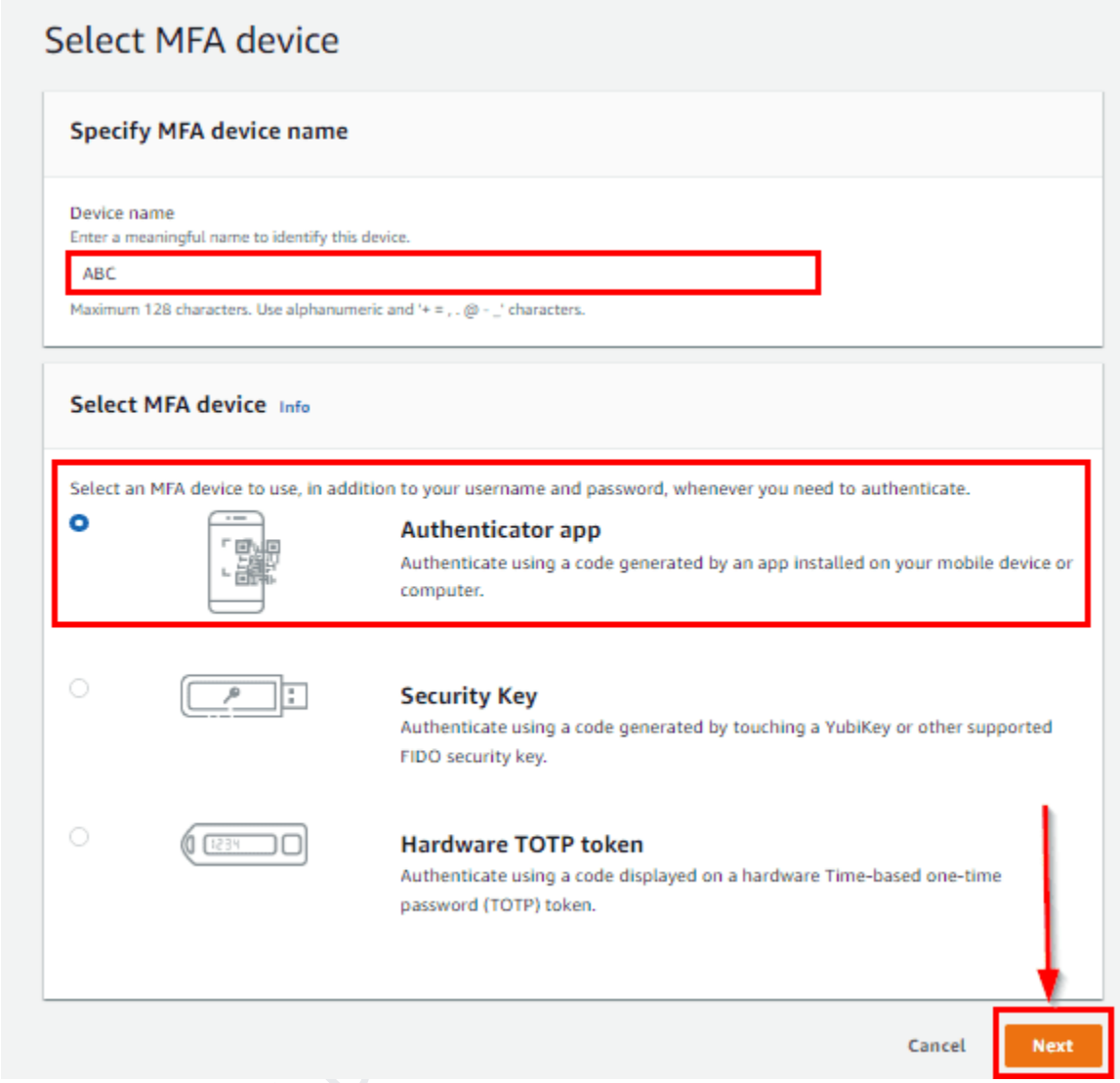
2) On the right side of the navigation bar, choose your account name, and choose My Security Credentials.



3) Click on Assign MFA device.



4) Choose Virtual MFA Device and click on Continue.

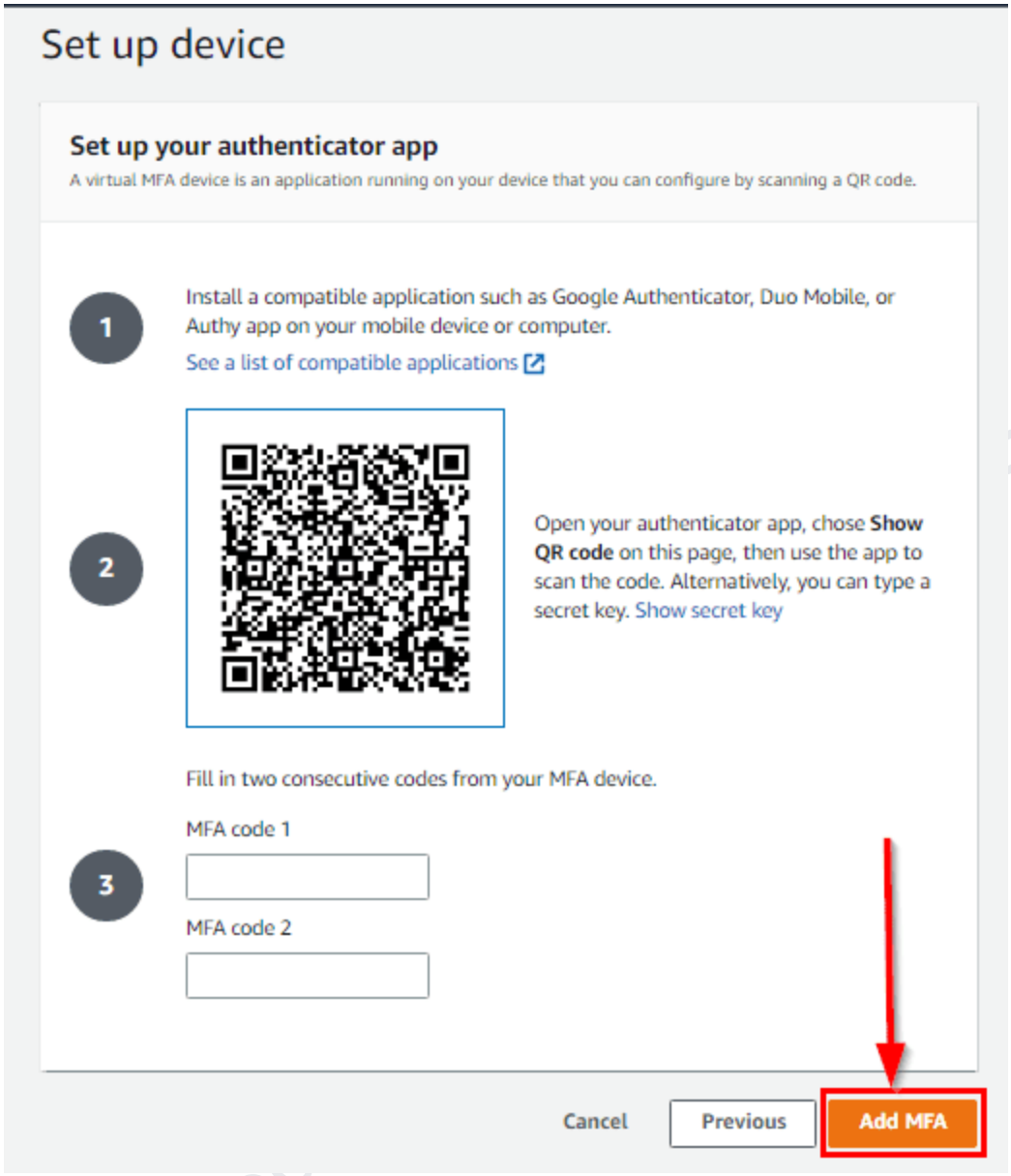


5) Now Install Google Authenticator on your phone.

Android: Click here

IOS: Click here

6) Now Click on Show QR Code and open the Google Authenticator app on your phone

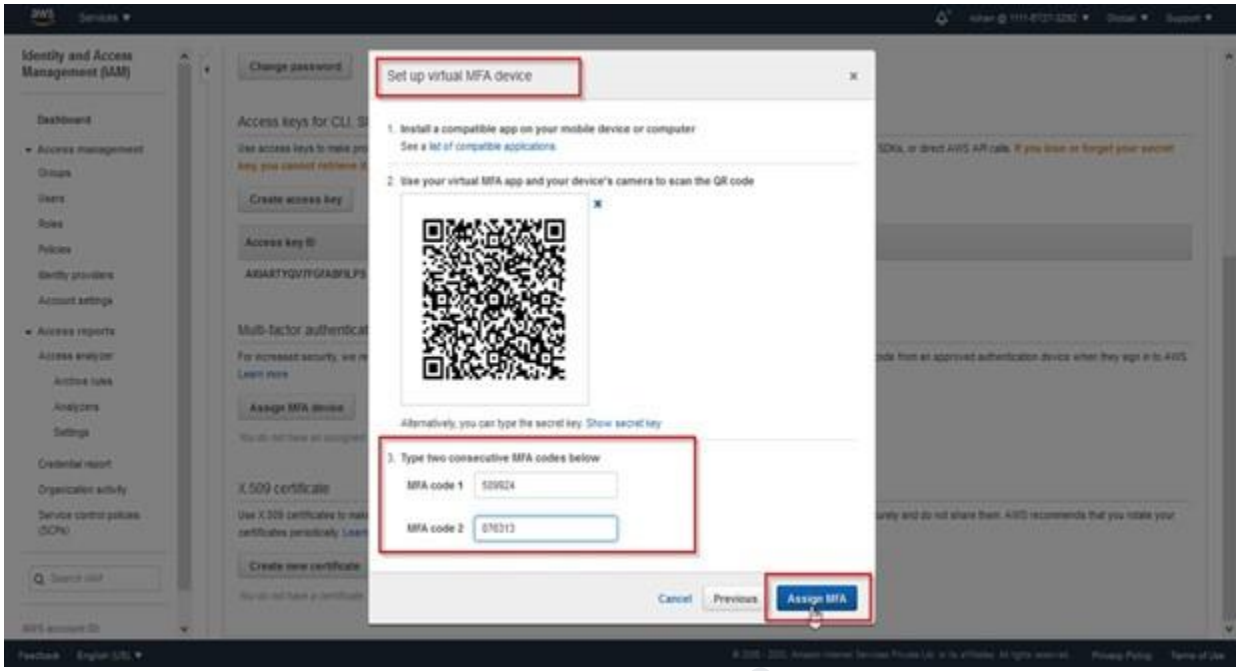


Note: Take a screenshot of the code so that in the future if you lose your phone you can use it to re-enable MFA

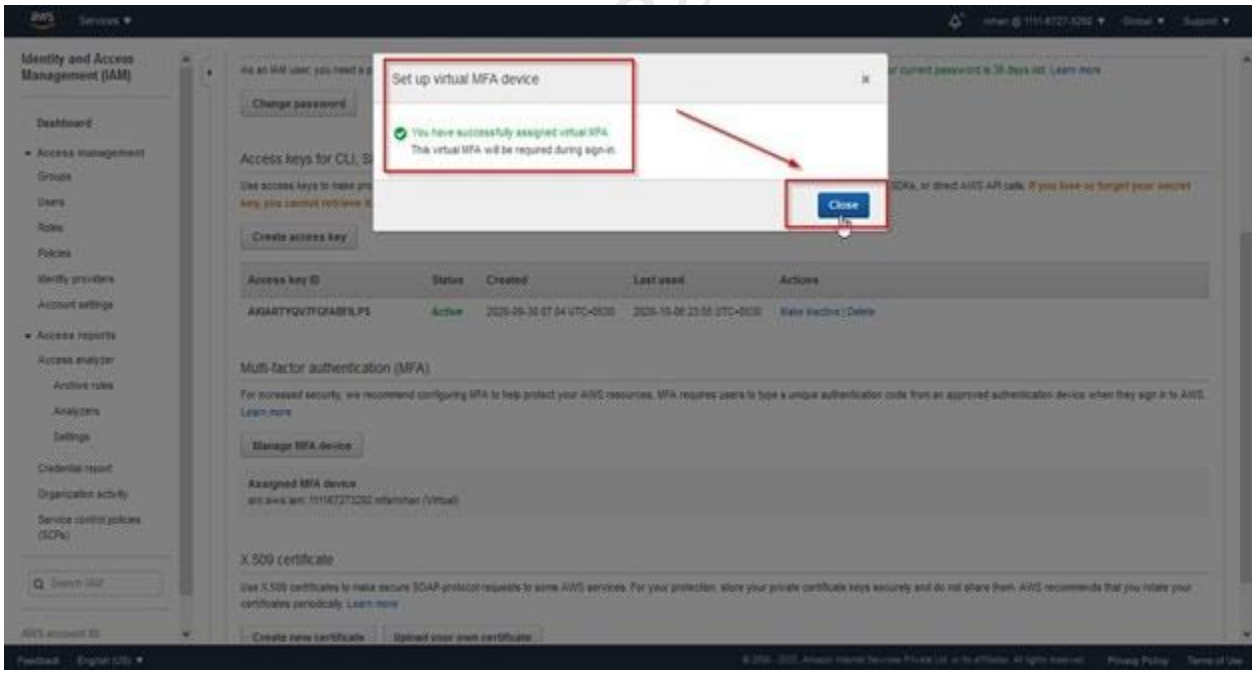
7) Now open the Google Authenticator App Click on Get started and Scan the QR code.

8) Now Enter the code from your Phone into MFA code 1 and MFA code 2.

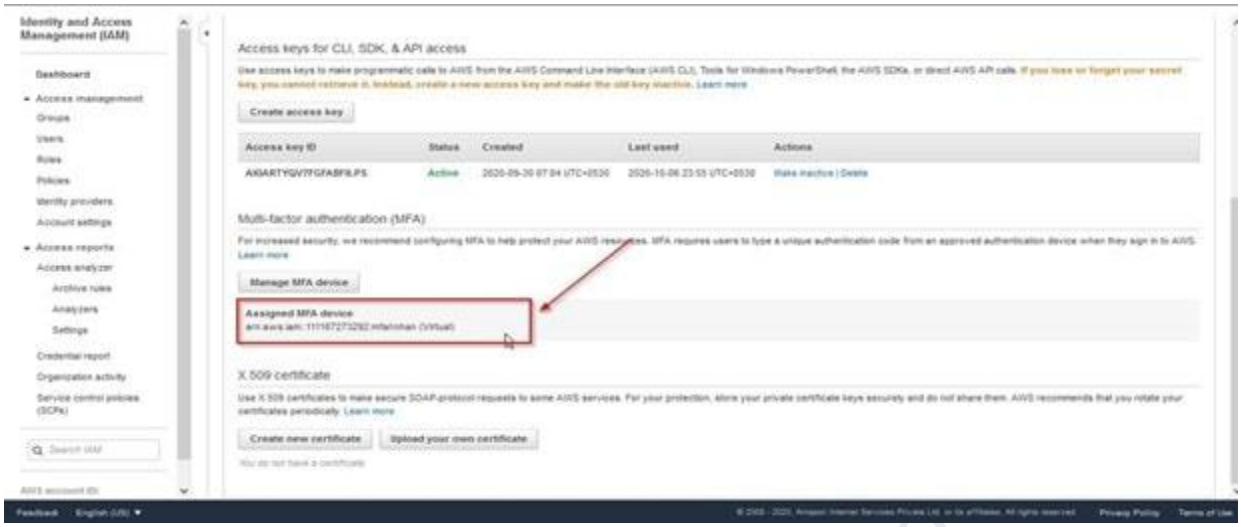
9) After adding MFA code click on Assign MFA



10) You will get a success message then click on Close



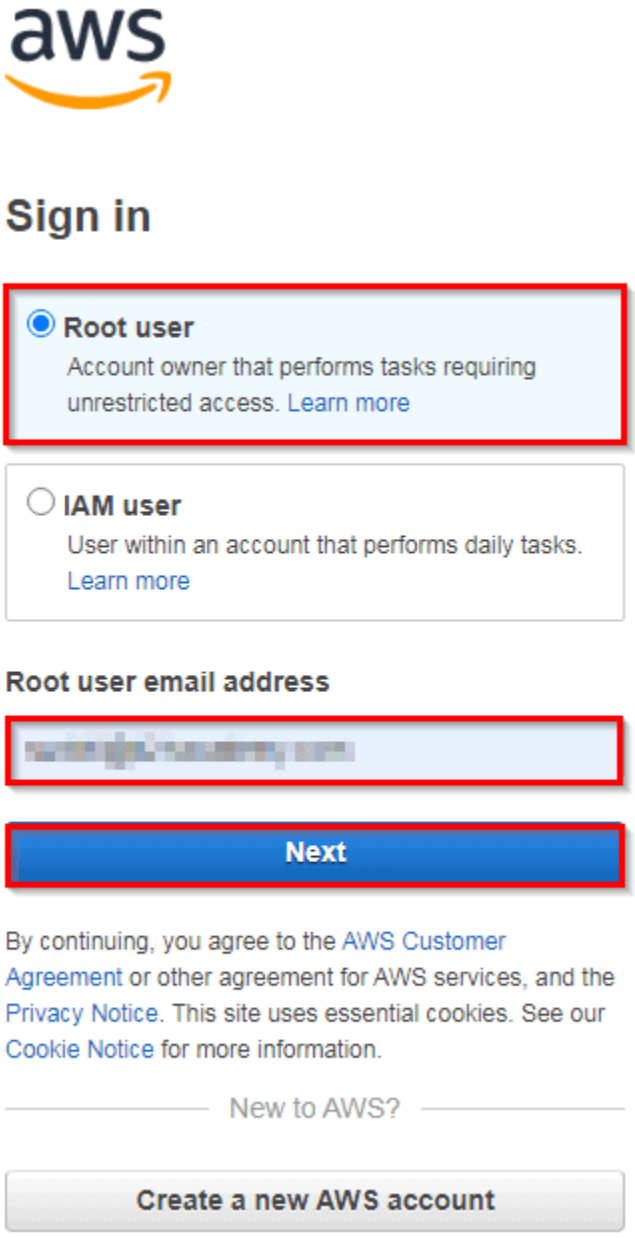
11) Now you will see that the device has been added for MFA



12) Now you have successfully Activated MFA on your root account setting

Accessing AWS Console Using MFA

1)Open your AWS console login page and click on Root User then enter your email



2) Enter your password corresponding to the Email address

3)Use your Google Authenticator Application on mobile and enter MFA code in AWS Console

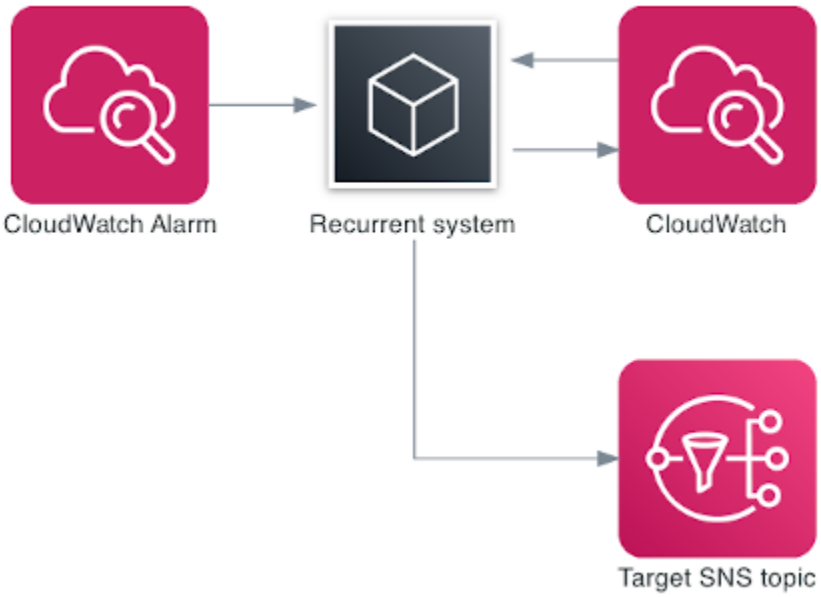
So this was an overview of AWS MFA and how you can enable it.

What if the MFA device does not work?

If your virtual MFA device or hardware MFA device appears to be functioning properly, but you cannot use it to access your AWS resources, it might be out of synchronization with AWS. For information about synchronizing a virtual MFA device or hardware MFA device, resynchronize your virtual and hardware MFA devices.

If your AWS account root user multi-factor authentication (MFA) device is lost, damaged, or not working, you can recover access to your account. IAM users must contact an administrator to deactivate the device.

#2 Task: Setup a CloudWatch Budgeting Alarms with SNS Alerts to make sure you stay notified if the expenses go beyond your threshold limit.



What are AWS Billing Alerts?

AWS billing alerts are enabled through Amazon CloudWatch, the AWS service dedicated to monitoring all activities across your AWS account.

In addition to billing alerts, CloudWatch provides infrastructure for monitoring applications, collecting metrics, logs, and other metadata, and detecting aberrant activity in your AWS usage.

AWS billing alerts are a subset of a more general product known as the AWS CloudWatch Alarms. AWS CloudWatch Alarms is a general system which allows you to set up notifications based on predefined events or activity within your AWS services and account(s). Cloudwatch provides a variety of metrics based on which you can schedule your alarms.

For example, you could create an alarm to notify you when the CPU Utilization of a running instance surpasses 85%. You can also create compound expressions to threshold metrics. For example, you could create an alarm that notifies you when your instance's CPU usage surpasses 85% AND your total monthly EC2 bill goes over $100.

Are AWS Billing Alerts Free? How Much Do They Cost?

As with everything AWS, billing alerts come at a marginal cost. AWS has a free tier that provides 10 alarms and 1,000 email notifications per month. This may be sufficient for smaller businesses.

For larger businesses, additional resources may be needed. In this case, you'll be looking at $0.10 per standard resolution alarm metric and $0.30 per high resolution alarm metric.

For the SNS portion of the service, you will incur a charge of $0.06 for every 100,000 HTTP notifications and $2.00 for every 100,000 email notifications.

How To Set Up AWS Billing Alerts

We'll now walk you through creating a billing alert so that you can familiarize yourself with the process.

Step 1: Go to the CloudWatch console
Navigate to https://console.aws.amazon.com/cloudwatch/

Step 2: Change the region
In the upper right hand corner, make sure your AWS region is set to US East (N. Virginia). This is the region where all billing data is stored.

Step 3: Create an alarm
Click on "Alarms" in the left side panel and in the dashboard that pops up click on the orange button that says "Create Alarm".

Step 4: Select a metric
In the panel that pops up, click the button that says "select metric". This will pop up a window that allows you to choose which type of service you'd like to use. Select "Billing".

Step 5: Select the service

In the next window, you can choose to either base the alarm off a given service's charges or the Total Estimated Charge on your account. We will set an alarm for our EC2 charges, so click "By Service".

On the next step, mark the checkbox next to the specific service(s) you'd like to monitor. We'll choose Amazon EC2.

Step 6: Select the time period
Next you'll want to set the time period that the alarm will be active for. You can choose anything from 10 seconds to an entire day. The interface will also show a graph which plots your current usage in blue against the alarm threshold in red. If the blue line crosses the red at any point during the selected time period, the alarm will activate.

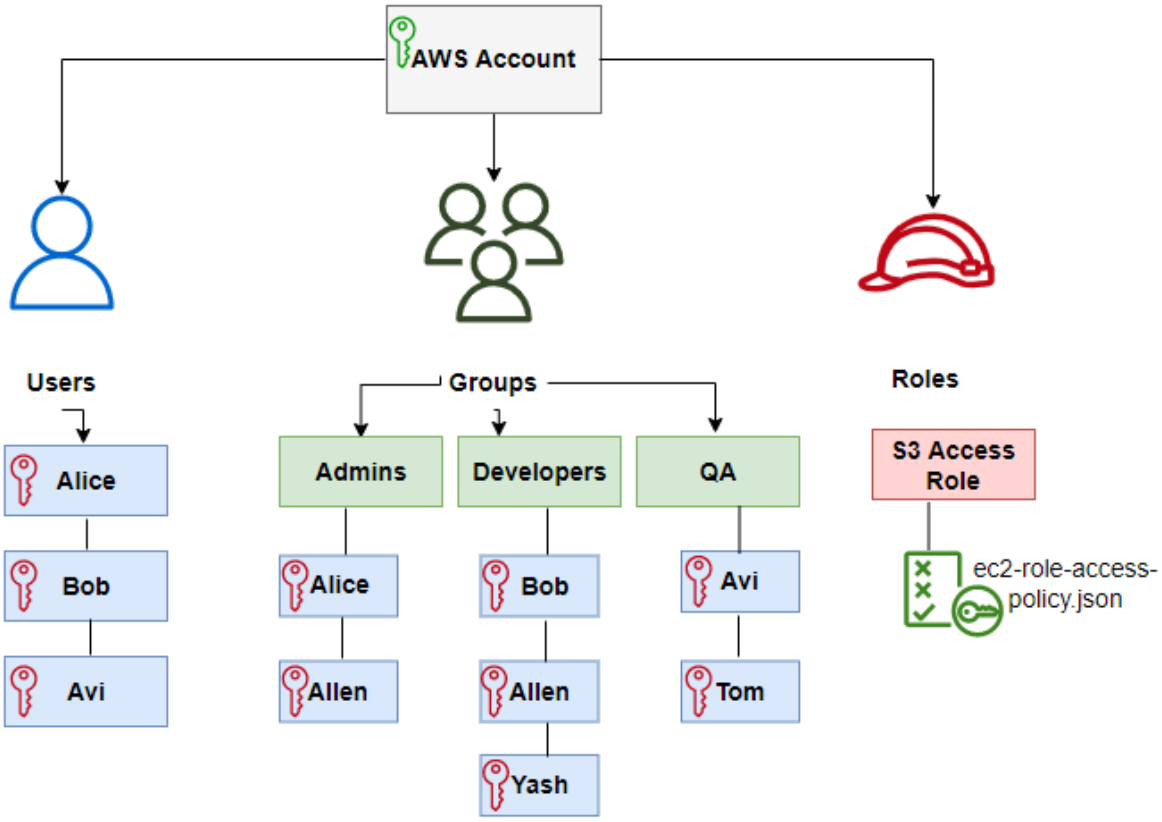
Step 7: Set the threshold
Next, set the threshold over or under which the alarm will ring. We'll set our alarm to activate if our EC2 charges exceed $10 within the next 6 hours. You can also use the anomaly detection panel to specify a band around your usage outside of which you'd like the alarm to activate.

Step 8: Create a notification
Next you'll want to create a notification using Amazon SNS. You just click "create new topic", give it a name, and enter the email you'd like to be notified at into the box.

Step 9: Add a description
Now you can create a name and description that will help you to remember what trigger is set for your alarm.

Step 10: Preview and create
A final page will now open which summarizes and displays your alarm's settings as you've created them. If everything looks good, you can click "Create Alarm" to have your alarm set.

#3 Task: AWS IAM Roles, Policies, User & Group management along with AWS CLI Operations.



Identity and Access Management (IAM) provides fine-grained access control across AWS accounts. With IAM, you can specify who can access which services and resources, and under which conditions. IAM is a pillar of security and provides you with easy ways to secure AWS accounts and resources.

IAM Key Points

- IAM controls access to AWS services and resources.
- IAM enables access between AWS services (e.g. EC2 to RDS).
- The main feature of IAM is that it allows you to create separate usernames and passwords for individual users or resources and delegate access.
- IAM supports identity federation. If the user is already authenticated, such as through a Facebook or Google account, IAM can be made to trust that authentication method and then allow access based on it.

- IAM is a free service. There is no additional charge for IAM security. There is no additional charge for creating additional users, groups, or policies.
- IAM is PCI DSS compliance.
- IAM supports MFA.

It provides two essential functions that work together:

Authentication:

- Validates the identity of a user or a service.
- Create and manage IAM identities.
- Federate corporate identities.

Authorization:

- Defines the permissions and limits access to only specific resources for the permitted user.
- Assign permissions to IAM identities.
- Assign permissions to corporate identities.

IAM Key Terms

IAM provides three primary types of identity: users, groups, roles. They serve different purposes, and each has an associated set of permissions using policies.

IAM User

- Users represent people (For example, members of the Development and DevOps team).
- Users are used to giving individuals the ability to manage AWS resources via AWS Console or programmatically (e.g. CLI).
- User is an identity with an associated credential and permissions attached to it. i.e. IAM User is an account for a single individual to access AWS resources.
- Users are granted permissions, either by attaching a permissions policy or by adding them to a group. Access appropriate resources through assigned permissions.
- User can be a Person or a Service.
- IAM User can belong to a max of 10 Groups.

- When an AWS account is first created, it has a single user. This is the root user, which has complete access to all resources.

IAM Account Root User Overview

IAM User Group

- Group is a collection of users and makes it easy to grant permissions to a class of users.
- You can logically group users in the AWS account and manage the privilege level for all users in that group.
- You set permissions for the group, and those permissions are automatically applied to all the users in the group.
- For example, Group for developers, Group for System administrators. Both of them will have different levels of access to AWS services.
- Group is not a true identity. You can leverage groups for easier administration.
- IAM users can be added to multiple groups.
- IAM policies can be attached to a group. Each group can have up to 10 policies attached.
- IAM users within an IAM Group inherit all policies attached to a group.
- Using groups is not only easier but also more secure and manageable.
- User group cannot be identified as a Principal in a resource-based policy. User group is a way to attach policies to multiple users at one time.

IAM Role

- Roles have many of the same properties as users. However, roles are not linked to a single individual.
- You can create roles to give permission to use AWS service by another AWS service.
- Role enables a user or AWS service to assume permissions for a task. Roles are assumed by IAM Users (People or Applications) and external (federated) users.
- Roles do not have log-in credentials. Instead, roles are temporarily assumed by users who need a specific set of permissions to complete a task. It delegates access to a trusted entity.

- For example, you can create S3Admin role and assign it to your EC2 instance. This will enable that EC2 instance to manage S3 resources.
- You can leverage roles for easier administration.
- Policies can be attached to Roles.
- Roles rely on the AWS STS service.
- You can assume an IAM role by using AWS Security Token Service (STS) operations or switch to a role in the AWS Management Console to receive a temporary role session.

You should never put keys into code or instances, use Roles instead.

IAM Policy

- Policies are JSON document that defines permission and controls access AWS resources. Permissions specify who has access to the resources and what actions they can perform.
- Policies are an authorization mechanism. You can create IAM policies to define granular access to resources.
- For example, a policy could allow an IAM user to access one of the buckets in Amazon S3.
- Policy can be attached to IAM identities (Users, Roles, Groups).
- Single policy can include multiple permissions (or statements).
- Policies can be either customer managed or managed by AWS.
- There are multiple policies provided by AWS like Administrator, S3FullAccess, etc.
- You can also create your own policy and use it to manage privilege levels to your AWS resources.

Policy contains the following information:

- Who can access it.
- What actions that user can take.
- Which AWS resources that user can access.
- When they can be accessed.

Types of policies:

- **Managed policies:** It is a default policy that you attach to multiple entities (users, groups, and roles) in the AWS account. Managed policies, whether

they are AWS-managed or customer-managed, are stand-alone identity-based policies attached to multiple users and/or groups.

- **Inline policies:** It is a policy that you create that is embedded directly into a single entity (user, group, or role).

Summary

IAM is one of the major building blocks of AWS. IAM lets you control who can use your resources (authentication) and in which ways (authorization). AWS IAM follows an incredibly granular approach in providing permissions and access control within your AWS environments.

Installing and Configuring the AWS CLI

You'll need to use the AWS CLI if you want to create or interact with an EMR Cluster using commands. Take these important steps to install and configure it.

The AWS Command Line Interface (AWS CLI) is a command-line tool that allows you to interact with AWS services using commands in your terminal/command prompt.

AWS CLI enables you to run commands to provision, configure, list, delete resources in the AWS cloud. Before you run any of the aws commands, you need to follow three steps:

1. Install AWS CLI
2. Create an IAM user with Administrator permissions
3. Configure the AWS CLI

Step 1. Install AWS CLI v2

Refer to the official AWS instructions to install/update AWS CLI (version 2) based on your underlying OS. You can verify the installation using the following command in your terminal (macOS)/cmd (Windows).

```
# Display the folder that contains the symlink to the aws cli tool
which aws
# See the current version
aws --version
```

See the sample output below. Note that the exact version of AWS CLI and Python may vary in your system.

Mac/Linux/Windows: Verify the successful installation of AWS CLI 2

Step 2. Create an IAM user (if you are using your own AWS account)

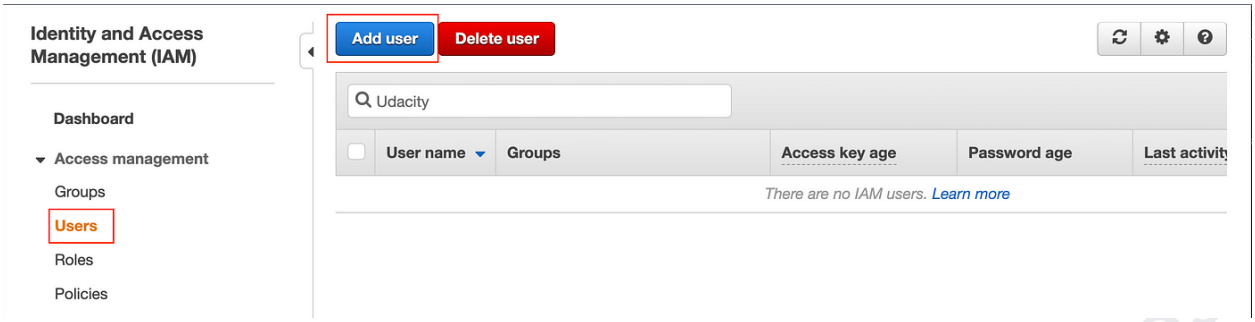
In this step, you will create an IAM user with Administrator permissions who is allowed to perform any action in your AWS account, only through CLI. After creating such an IAM user, we will use its Access key (long-term credentials) to configure the AWS CLI locally.

Let's create an AWS IAM user, and copy its Access key.

AWS Identity and Access Management (IAM) service allows you to authorize users / applications (such as AWS CLI) to access AWS resources.

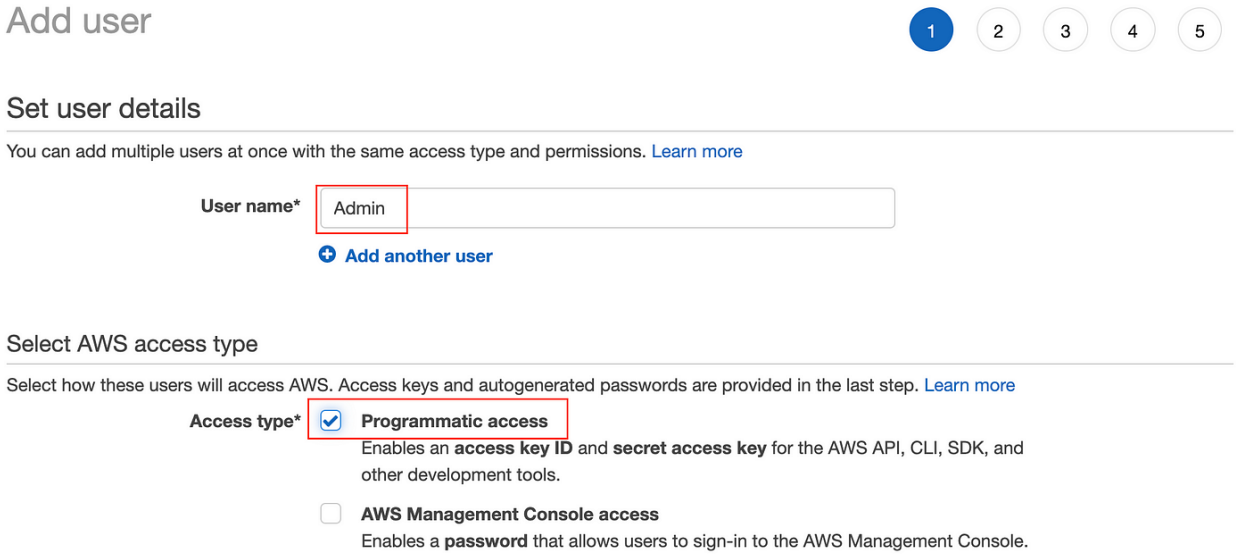
The Access key is a combination of an Access Key ID and a Secret Access Key. Let's see the steps to create an IAM user, and generate its Access key.

- Navigate to the IAM Dashboard, and create an IAM user.



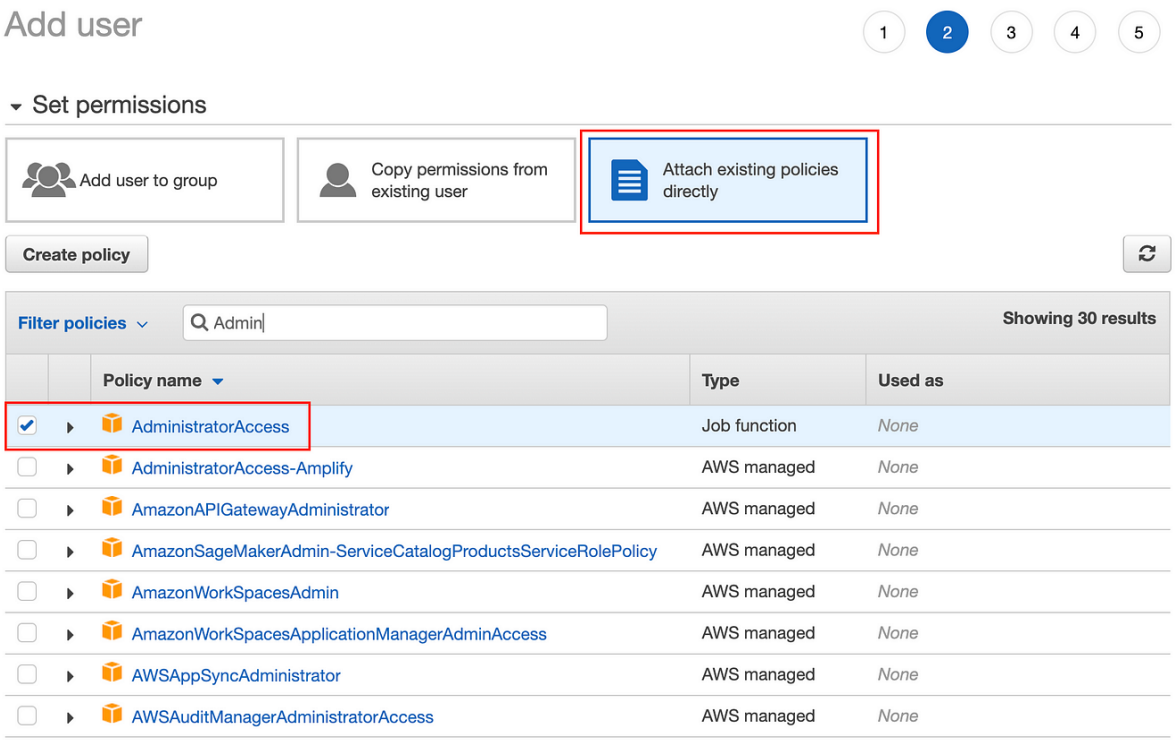
Add a new IAM user

- Set the user details, such as the name, and access type as Programmatic access only.



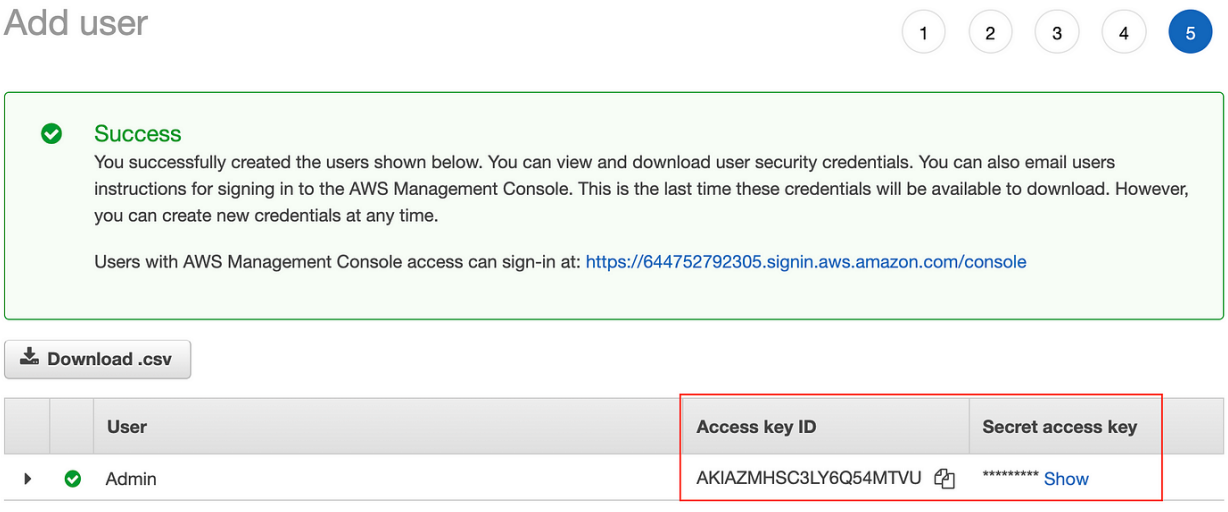
Set the user name, and type (mode) of access

- Set the permissions to the new user by attaching the AWS Managed AdministratorAccess policy from the list of existing policies.



Attach the AdministratorAccess policy from the list of pre-created policies

- Provide tags [optional], review the details of the new user, and finally create the new user.
- After a user is created successfully, download the access key file (.csv) containing the Access Key ID and a Secret Access Key. You can even copy the keys and stay on the same page. Don't skip this step as this will be your only opportunity to download the secret access key file.



Copy the Access key of the new user OR download the .csv file containing the Access key

Step 3. Configure the AWS CLI

You will need to configure the following four items on your local machine before you can interact with any of the AWS services:

1. Access key — It is a combination of an Access Key ID and a Secret Access Key. Together, they are referred to as Access key. You can generate an Access key from the AWS IAM service, and specify the level of permissions (authorization) with the help of IAM Roles. If you are using the Udacity-provided AWS Gateway and account, these credentials are provided in the popup that appears when you click “Launch AWS Gateway” in the course menu to the left.
2. Default AWS Region — It specifies the AWS Region where you want to send your requests by default.
3. Default output format — It specifies how the results are formatted. It can either be a json, yaml, text, or a table.
4. Profile — A collection of settings is called a profile. The default profile name is default, however, you can create a new profile using the aws configure --profile new_name command. A sample command is given below.
5. Session Token — If you are using the Udacity-provided AWS Gateway and Account, you will also need to add the session token from the AWS Gateway credential popup, as well as the Access Key and SecretKey from the popup as described here.

```
(base) xyz$ aws configure list
      Name                    Value             Type    Location
      ----                    -----             ----    --------
   profile                <not set>             None    None
access_key                <not set>             None    None
secret_key                <not set>             None    None
    region                us-east-2      config-file    /Users/xyz/.aws/config
(base) xyz$ aws configure --profile default
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [us-east-2]: us-east-2
Default output format [json]: json
(base) xyz$
(base) xyz$
```

- Set the default profile credentials

```
# Navigate to the home directory
cd ~
# If you do not use the profile-name, a default profile will be created fo
aws configure --profile <profile-name>
# View the current configuration
aws configure list --profile <profile-name>
# View all existing profile names
aws configure list-profiles
# In case, you want to change the region in a given profile
# aws configure set <parameter> <value>  --profile <profile-name>
aws configure set region us-east-1  --profile <profile-name>
```

Moving forward, you can use --profile <profile-name> option with any AWS command. This will resolve the conflict if you have multiple profiles set up locally.

The command above will store the access key in a default file ~/.aws/credentials and store the profile in the ~/.aws/config file.

Let the system know that your sensitive information is residing in the .aws folder

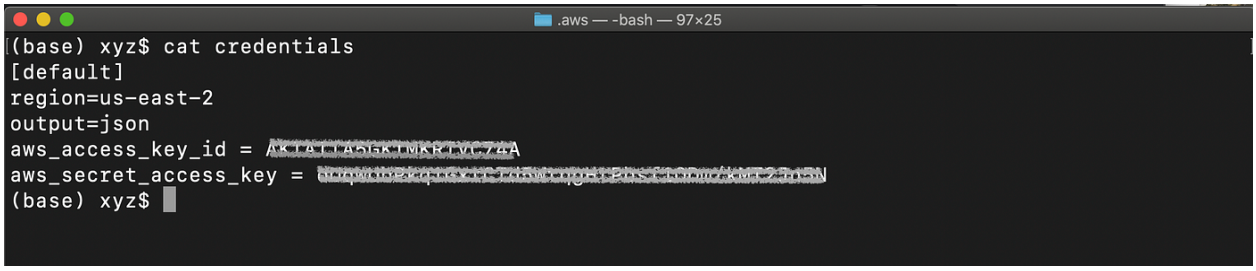
```
export AWS_CONFIG_FILE=~/.aws/config
export AWS_SHARED_CREDENTIALS_FILE=~/.aws/credentials
```

```
(base) xyz$ aws configure list
      Name                    Value             Type    Location
      ----                    -----             ----    --------
   profile                <not set>             None    None
access_key     ****************C74A shared-credentials-file
secret_key     ****************Jn5N shared-credentials-file
    region                us-east-2      config-file    /Users/xyz/.aws/config
(base) xyz$
```

Mac/Linux: A successful configuration

Setting the Session Token

After a successful credential set-up, your “credentials” file will look like this one below. If you are using the Udacity-provided AWS account, you should have a session token name/value pair in your configuration as well.



Mac/Linux: View the credentials file using cat ~/.aws/credentials command

Step 4. Run your first AWS CLI command

- Check the successful configuration of the AWS CLI, by running either of the following AWS command:

```
# If you've just one profile set locally
aws iam list-users
# If you've multiple profiles set locally
aws iam list-users --profile <profile-name>
```

The output will display the details of the recently created user:

```
{
"Users": [
    {
        "Path": "/",
        "UserName": "Admin",
        "UserId": "AIDAZMXYZ3LY2BNC5ZM5E",
        "Arn": "arn:aws:iam::388752792305:user/Admin",
        "CreateDate": "2021-01-28T13:44:15+00:00"
    }
]
}
```

Troubleshoot

If you are facing issues while following the commands above, refer to the detailed instructions here -

1. Configuration basics
2. Configuration and credential file settings
3. Environment variables to configure the AWS CLI
4. Using the Session Token

Updating the specific variable in the configuration

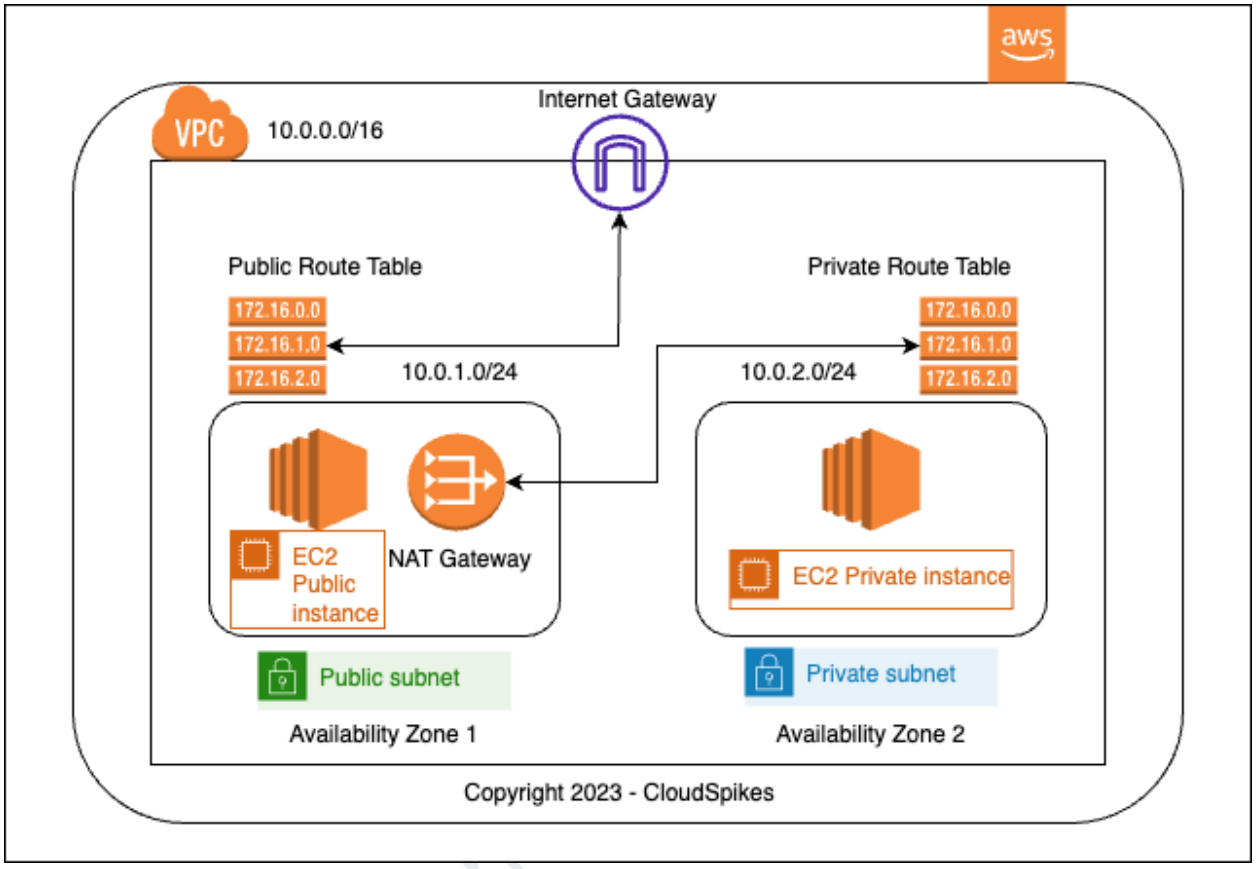
In the future, you can set a single value, by using the command, such as:

```
# Syntax
# aws configure set <varname> <value> [--profile profile-name]

aws configure set default.region us-east-2
```

It will update only the region variable in the existing default profile.

#4 Task: Create VPC, Public & Private Subnets.



What is a VPC and why is it important?

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

Amazon VPC enables you to build a virtual network in the AWS cloud — no VPNs, hardware, or physical data centers required. You can define your own network space, and control how your network and the Amazon EC2 resources inside your network are exposed to the Internet.

What is the importance of having private and public subnets?

The instances in the public subnet can send outbound traffic directly to the internet, whereas the instances in the private subnet can't. Instead, the instances in the private subnet can access the internet by using a network address translation (NAT) gateway that resides in the public subnet.

Requirements:

- AWS Console → https://aws.amazon.com/free/
- Patience…

The Process:

- Follow the steps below. Use the same IPv4 CIDR block numbers. You can make the names of the VPC, subnets, route tables, IGW, NAT gateways unique to whatever you want.

1. Create VPC from VPC Dashboard from AWS Console.

- Click on Create button which will create a VPC

2. Create subnets now, we'll start with creating a public subnet now.

VPC ID is the vpc you created

Availability Zone I left as no preference for this particular project.

3. Let's create a private subnet now.

This can be created using Subnets options from left hand side list in VPC Dashboard. Same process as the public subnet, except we are using a different IPv4 CIDR block.

- Click on create subnet button.

What is an AWS CIDR block?

- When you create a VPC, you must specify a range of IPv4 addresses for the VPC in the form of a Classless Inter-Domain Routing (CIDR) block; for example, 10.0. 0.0/16 . This is the primary CIDR block for your VPC.

4. Modify Auto assign IP by right clicking on public subnet.

5. Create an Internet Gateway to use with our Public Subnet.

Yes, it's really as easy as just creating a tag.

6. Attach Internet Gateway to VPC.

- When you right click on internet gateway, it will show you Attach to VPC option as below.
- Select the VPC you created and click on attach.

7. Create Route Tables.

- A route table contains a set of rules, called routes, that are used to determine where network traffic from your subnet or gateway is directed. To put it simply, a route table tells network packets which way they need to go to get to their destination.

public route table

private route table

- Add Internet Gateway to Public Route Table. Click ADD routes and attach.

8. Edit Subnet Association

- Repeat steps for both your public and private Route Tables.
- Click Edit Subnet Association button.

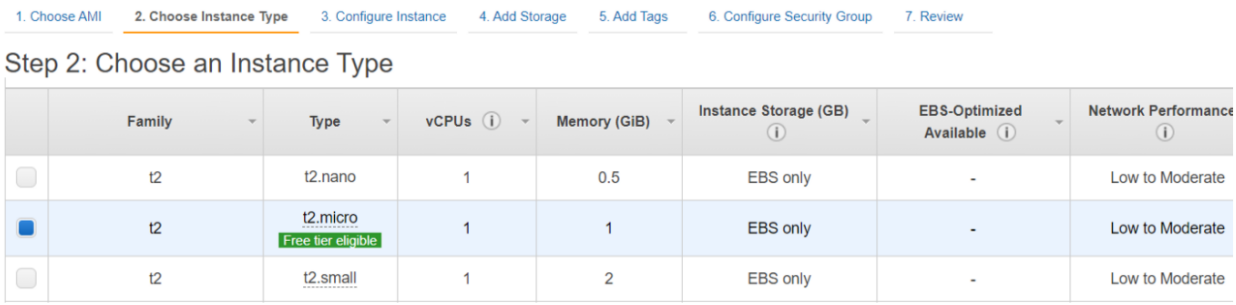
public route table
private route table

9. Create public and private EC2 instances.

- Follow process for both public and private instances.
- Pay attention to steps for what is particular to a certain instance.

- First step, choose an AMI.

- Instance Type.



- Public Configure Details

public instance

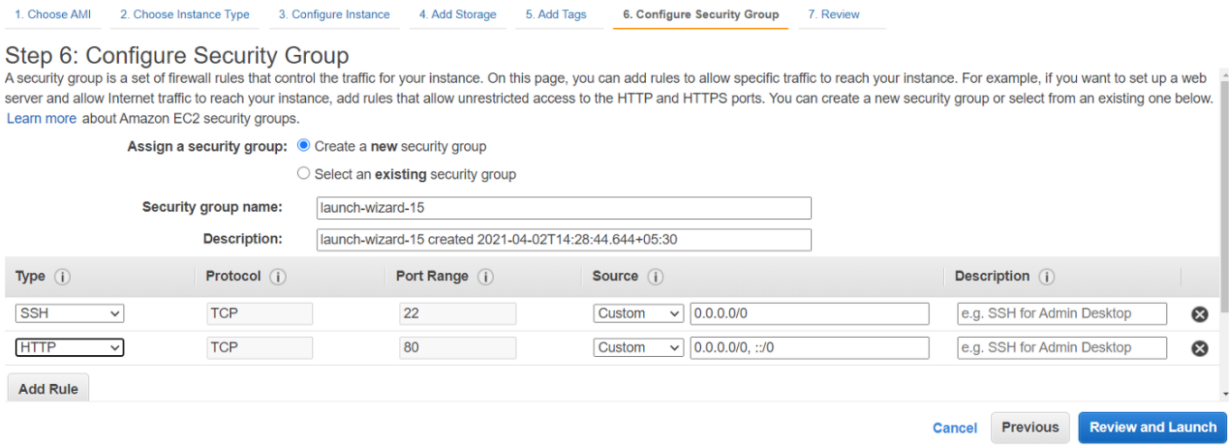
- Private Configure Details

configure instance details should be the same for both instance, except the public and private subnets.

- Copy and paste User Data from below into PUBLIC instance.

```
#!/bin/bash
yum install httpd –y
yum update -y
service httpd start
chkconfig httpd on
```

- Configure Security Group. Make sure to add HTTP port 80 to both public and private instances. SSH port 22 will already be there when created.



configure security group is the same for both public and private instances

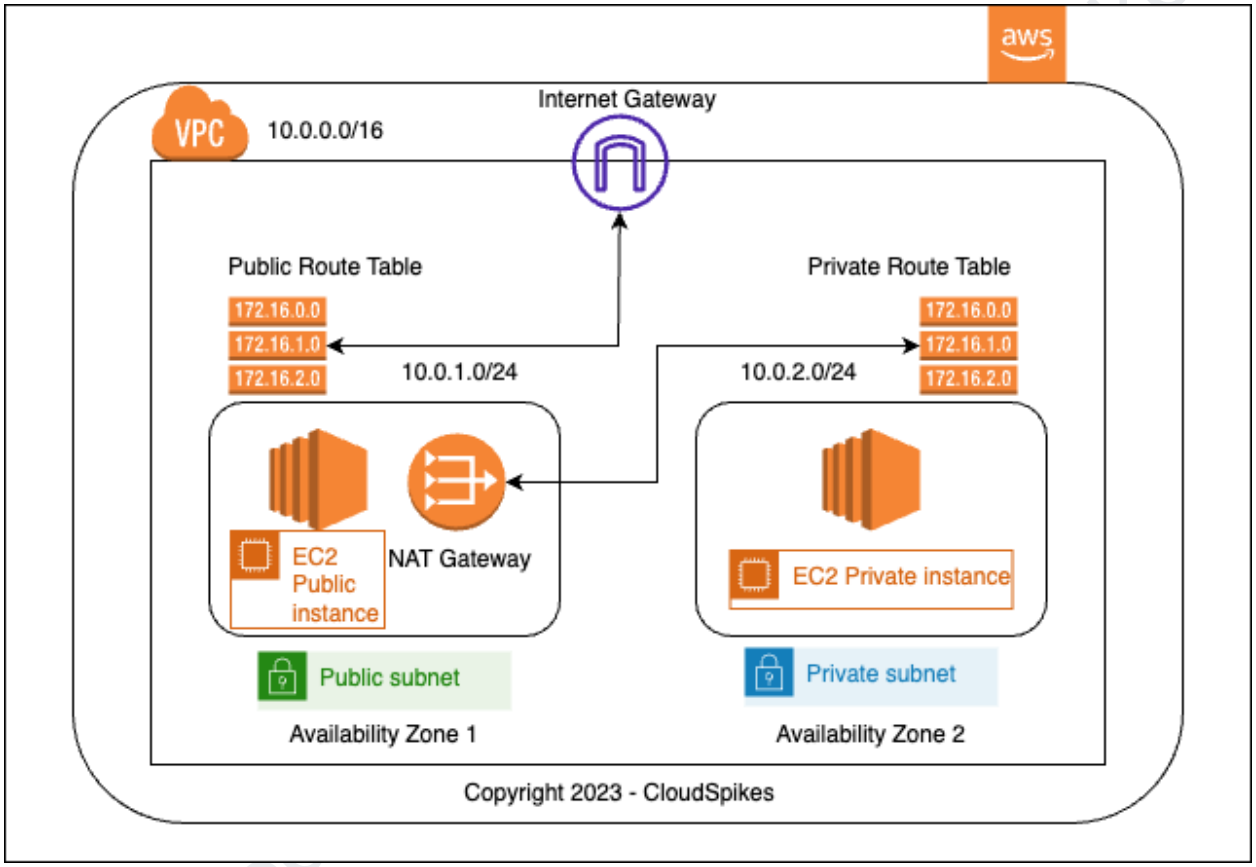
public and private instances should be running as so

- Create a keypair or using an existing keypair.

10. Configure NAT Gateway with it's private route table mapping.

- Create a NAT Gateway in the public subnet*. Ensure you are creating the NAT Gateway in public subnet only.
- Attach an Elastic IP Address* to the NAT Gateway. Now a routing should be added to your private subnet
- Go to private subnet's routing table. Add a route to internet through NAT
- Destination as 0.0.0.0/0 and Target as NAT gateway

#5 Task: Create Set up AWS EC2 Instances in Pub & Private Subnets. Expose Website hosted on Private EC2 instance via Public EC2 instance via Nginx or Apache Web Server.



1. Testing the EC2 Instances in the recently created VPC Network configurations.

Public Instance:

- right click on box next to name of instance and click connect

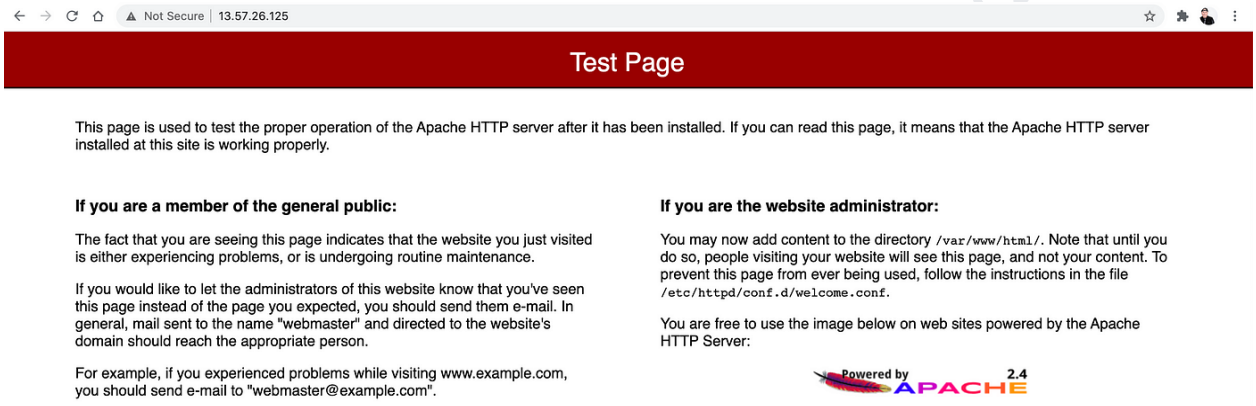
ssh steps to connecting to public instance

```
The authenticity of host '13.57.26.125 (13.57.26.125)' can't be established.
ECDSA key fingerprint is SHA256:2xIX63IFcwyXC62HpATCYhUim/duOvISzaGXo5PuP7E.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '13.57.26.125' (ECDSA) to the list of known hosts.
Last login: Sun May  9 06:44:18 2021 from c-73-241-240-172.hsd1.ca.comcast.net

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-1-233 ~]$
```

terminal of successfully connecting public instance



apache test page from our public instance IP address

Private Instance:

- We'll now connect to our private instance through our public instance.
- Inside you public instance create a file for your keypair.
- create a file for your keypair.

yum install vim
vim keypair.pem
:wq

- Copy and paste contents of keypair inside your newly created keypair.pem file, your keypair file will look like the following below.

chmod 400 keypair.pem

ssh -i keypair.pem ec2-user@private-ip-address

2. Conclusion.

- In conclusion, the machines on a private subnet can access the Internet because the default route on a private subnet is not the VPC “Internet Gateway” object — it is an EC2 instance configured as a NAT instance. A NAT instance is an instance on a public subnet with a public IP, and specific configuration.

Steps to setup Apache Web Server HTTPd on Ubuntu 22.04:

$ sudo apt update
$ sudo apt install apache2 -y
$ sudo systemctl status apache2

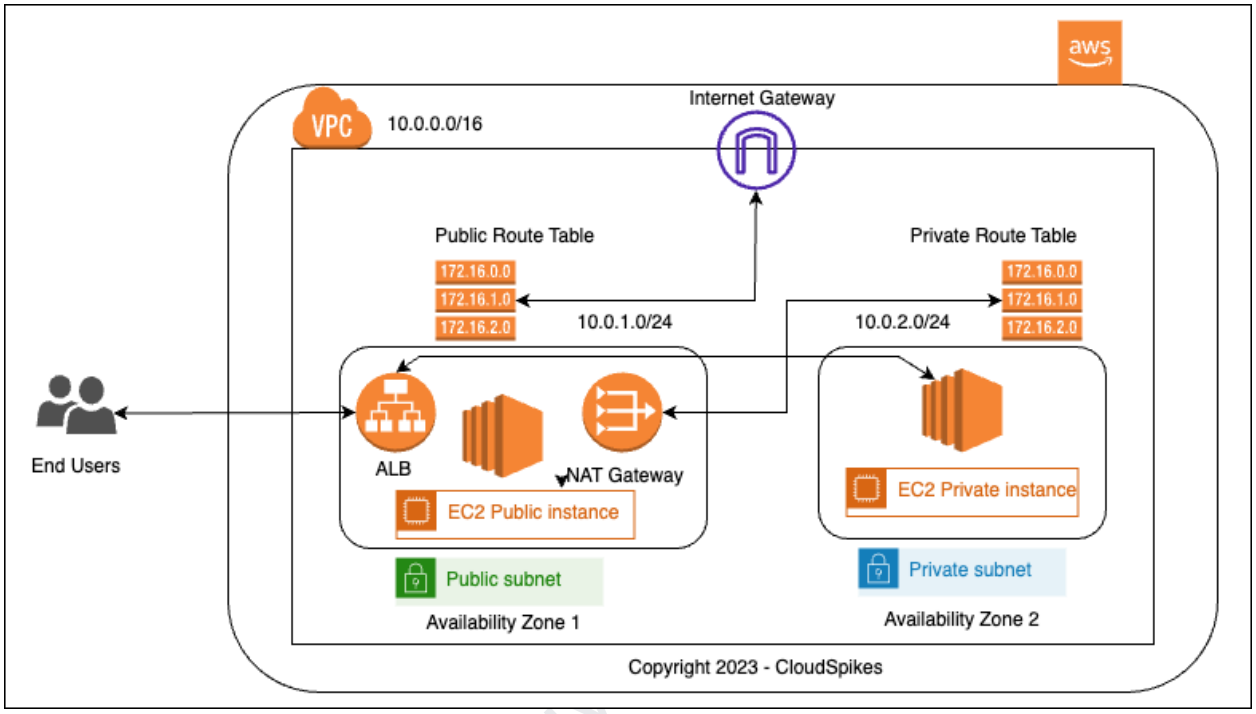
Steps to setup Nginx Web Server on Ubuntu 22.04:

$ sudo apt update
$ sudo apt install nginx -y
$ systemctl status nginx

Once Nginx/Apache is setup, you can replace default index.html with your website specific index file.

To do so, first you need to SCP the website template folder to the EC2 server and then replace the default Nginx/Apache index.html with the index file of your website along with it’s supporting files/folders such as CSS, JavaScript, Media files, etc.

#6 Task: Configure ALB from Public Subnet to access the Private Subnet website.



1. Create a Target Group under ALB section and point it to the Private EC2 instance.

2. Create an ALB with the Listners having Target Group created above.

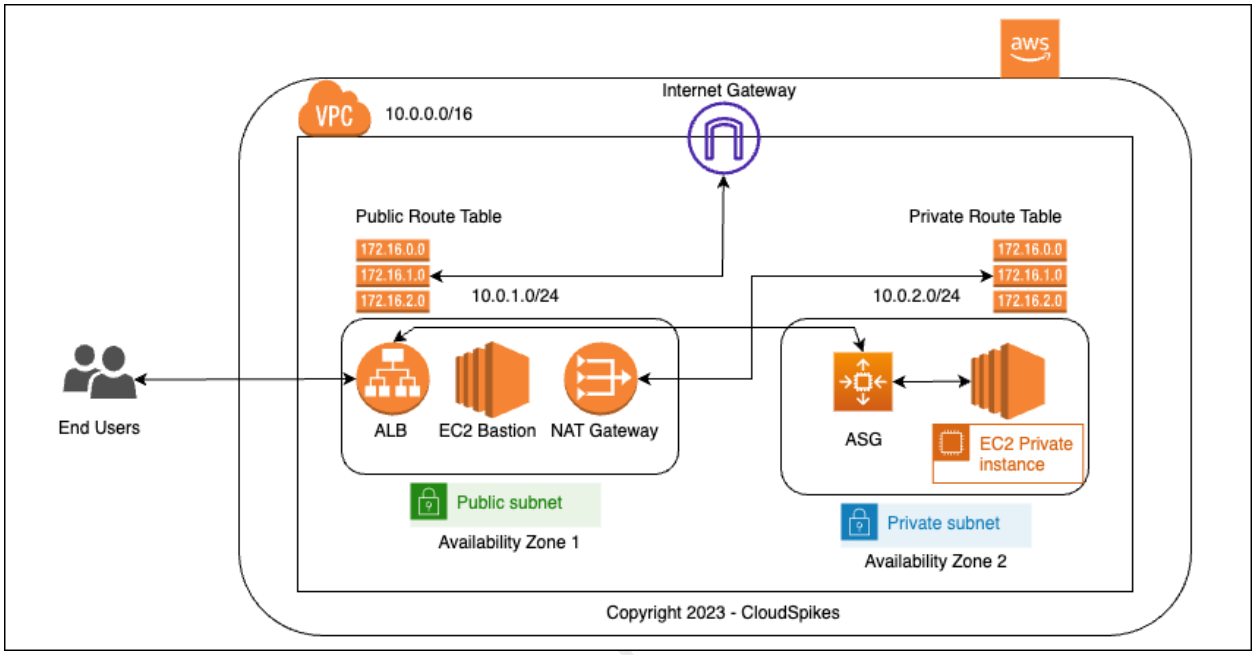
- Make sure you have atleast two public subnet to allocated the ALB with internet facing placement.
- Both of these subnets should have its respective route table association having internet gateway routes.

3. Also, create or reuse an existing Security Group to whitelist the App/Website Port (in this case it is 80).

You must add the private instance security group as well in the ALB security group selection. Here, you can add maximum up to five security groups.

4. You can now access your App/Website by calling the DNS Address of the provisioned ALB.

#7 Task: Define Launch Template for your Private EC2 Instance & configure ASG using it.



1. Create an EC2 Launch Template.

Create an EC2 Launch Template with a few basic details such as:

- **AMI (Ubuntu - ami-007855ac798b5175e)**,
- Select instance type as **t2.micro** to save cost,
- Select an existing **Key Pair** or create a new one,
- Select the network configs by selecting the created **VPC and its private subnet**.

2. Create an ASG using the Launch Template created before.

Go to Auto Scaling Group and create an ASG with the below configs:

- Provide the name and choose launch template created in step #1.
- Select the network configs by selecting the created **VPC and its private subnet**.
- Attach a new ALB or you can select an existing ALB.
- You can configure the ALB similarly how we did in Lab work #6.
- Go next and keep the default settings as is till notifications tab.

- Now, you can set the min, max and desired instance count which will set your ASG Scaling abilities based on the metric configs you provide i.e., Scaling policy set to **50% of the Avg CPU Utilisation** or any other metrics based on which you want to scale your application based on the application needs.
- In the Notifications tab, you can select an existing or create a new SNS topic with all the ASG event selection to keep the subscribed users of the SNS topic updated about any change in the ASG configs.
- Attach tags, review all the settings and hit create button at the end.

3. Wait till the instances comes up healthy as per the min/desired instance count.

Keep an eye on the ALB Target Group to identify the correct verification time when the ALB targeted EC2 instance which are spun via ASG configs comes up healthy, running and SSH reachable.

Once, the EC2 instance is SSH reachable, you can setup an Apache or Nginx web server as per the steps given in Lab work #5. Finally, after checking Aapche/Nginx service status from the EC2 bash/shell/terminal you can now hit the ALB DNS name (A record) to verify the default Apache/Nginx web page.

#8 Task: Deploy a static website on the S3 bucket to demonstrate a serverless approach.

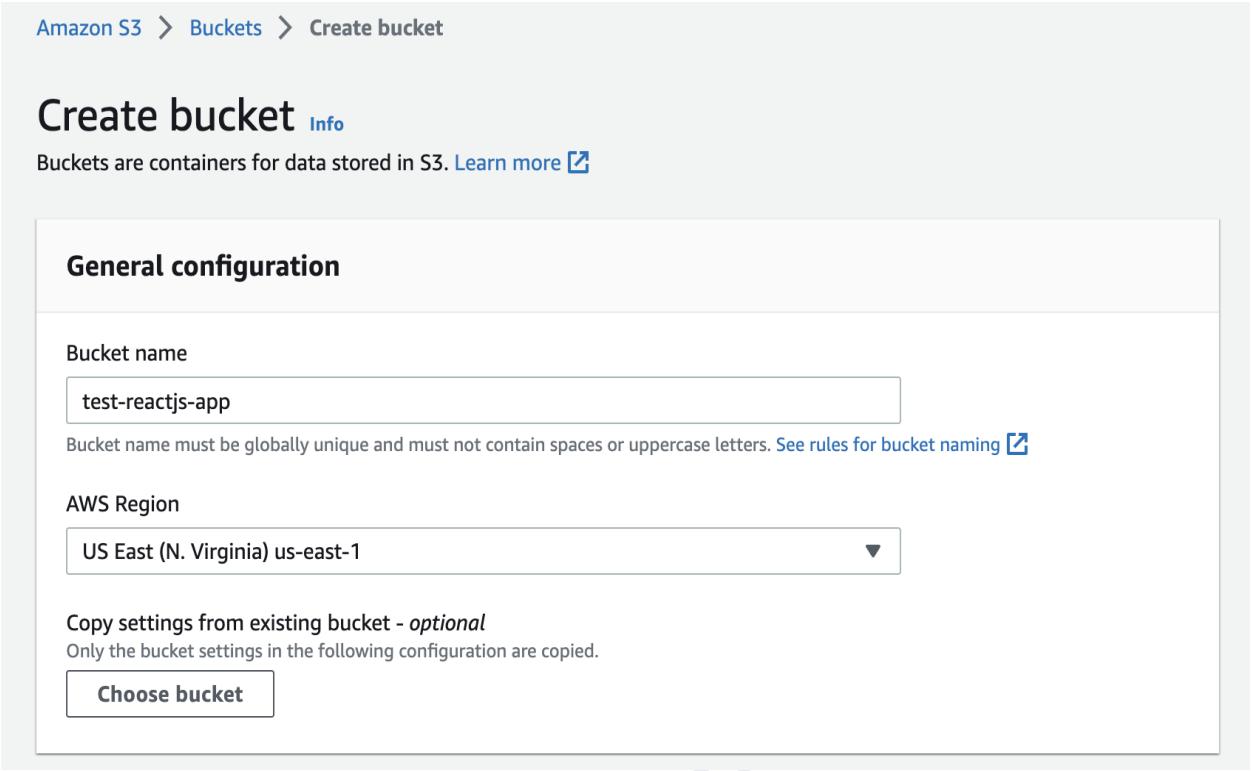
1. Setup all the prerequisites for ReactJs App.

- Make sure you have an AWS IAM User with required S3 access to deploy the ReactJs App.
- Configure AWS CLI on the machine where you want to deploy the ReactJs App using AK, SK and Region of your choice.
- Install Node, NPM and Yarn with the LTS (Long Term Support) version.

2. Start creating the S3 bucket for hosting your ReactJs Web App.

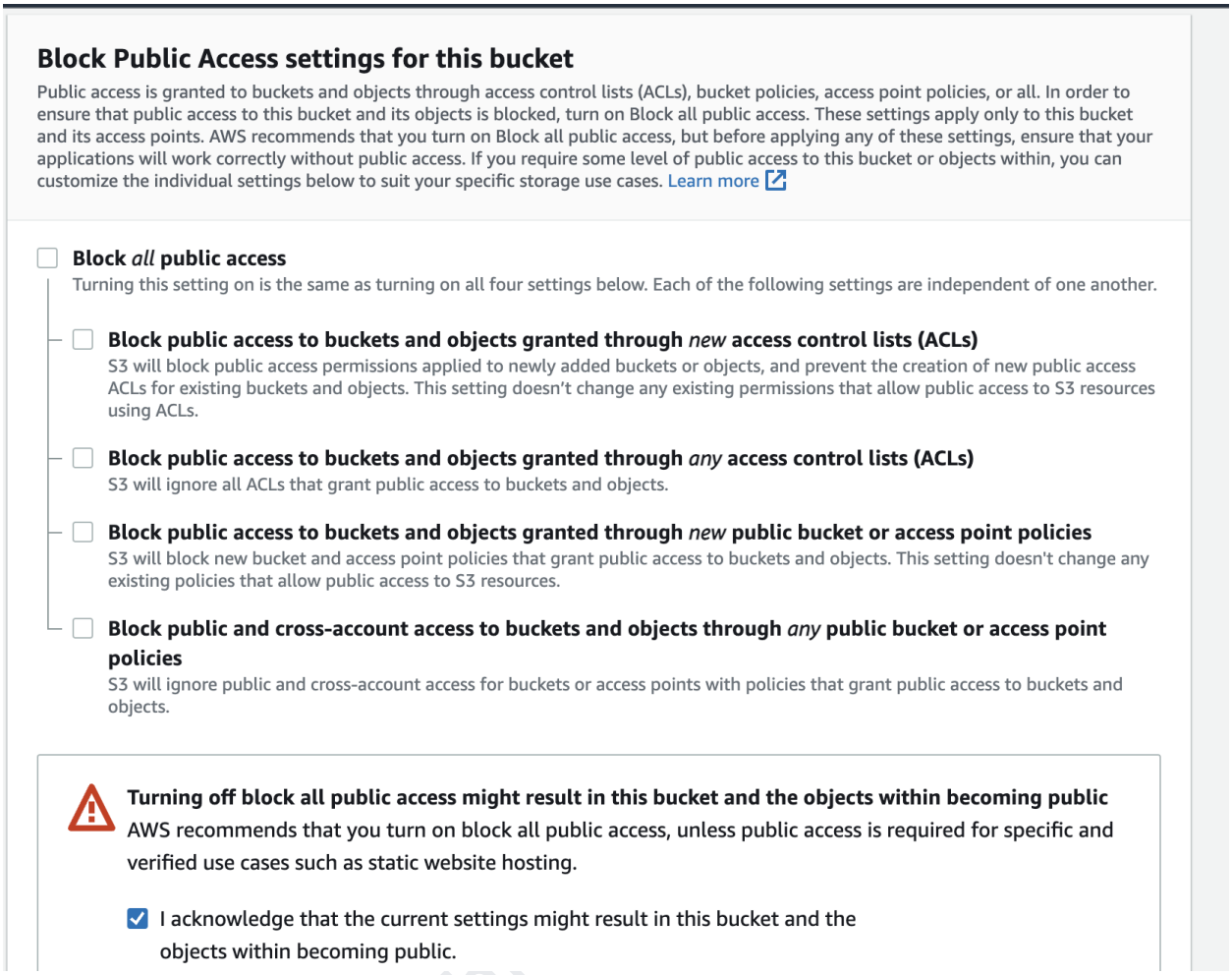
Follow the steps given below to setup a new S3 bucket:

- Go to S3 service in AWS Console and click on **Create bucket**.

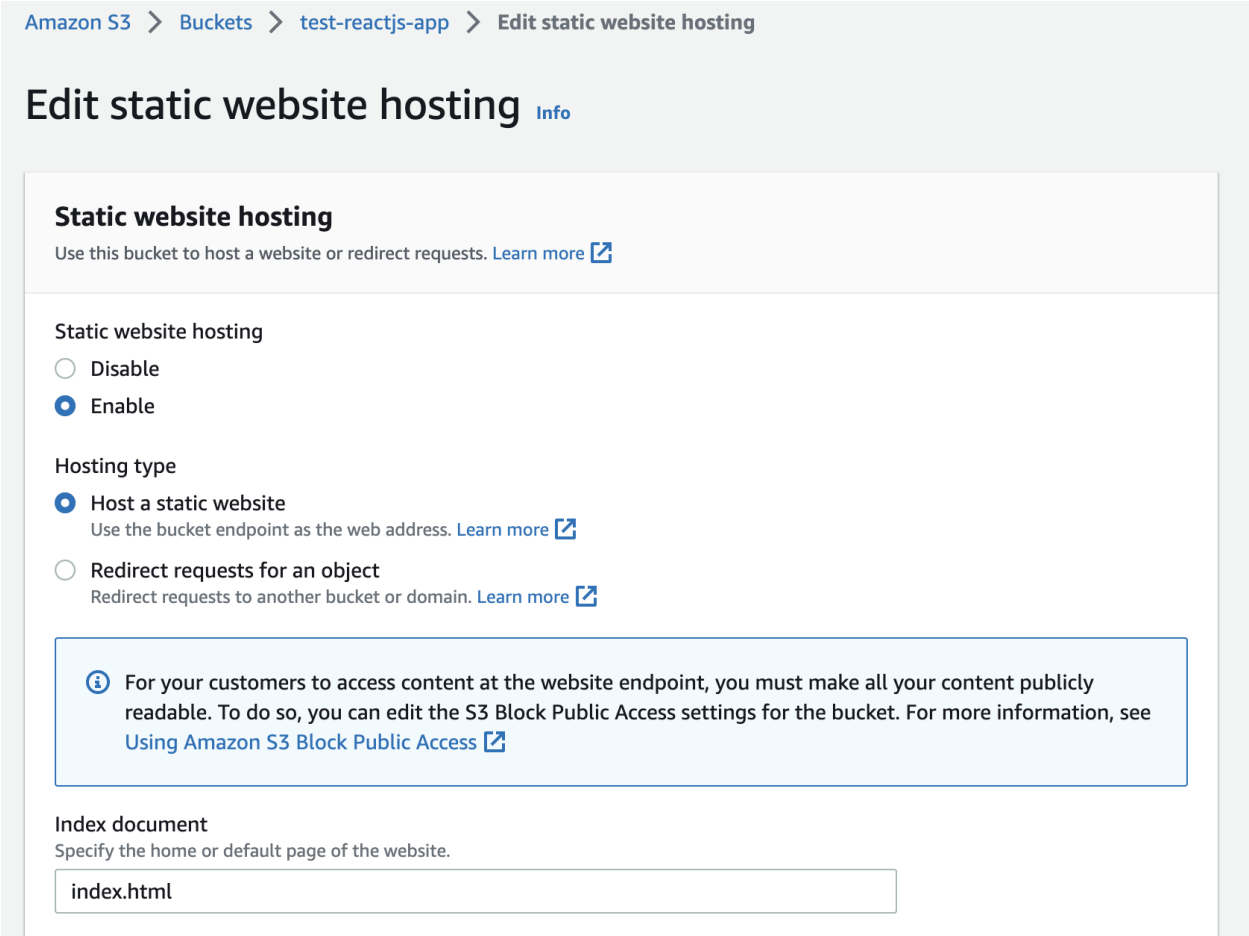


Note: AWS S3 bucket names are unique as it is a global service. Hence, **make sure you choose a unique non-existing S3 bucket name** and **replace it everywhere** as given in this document where **test-reactjs-app** S3 bucket name is referred.

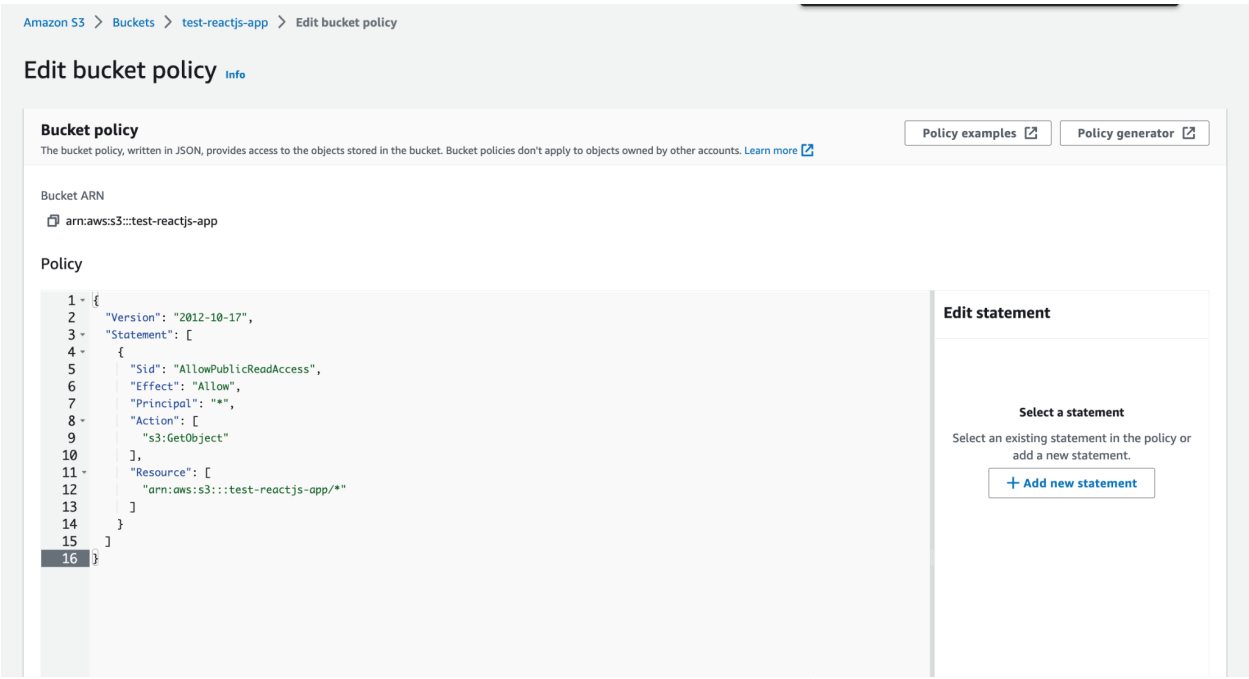
- Make sure you Allow Public access to let user access the ReactJs App deployed on the S3 bucket.



- Skip all the settings as default and hit **Create bucket**.
- Go to the newly created S3 bucket and **Properties** tab.
- Click on the **Static Website Hosting** button and enter **index.html** value in the **Index document** textbox.



- Now, go to the **Permissions** tab under the same bucket and select the **Bucket Policy** to enter the following **JSON Access configurations**.



- Save the policy document and you are done with the AWS S3 setup. Below is the JSON policy given where you just need to change the **YOUR_BUCKET_NAME_HERE** value with the bucket name you created a few moments ago:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "PublicReadGetObject",
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::YOUR_BUCKET_NAME_HERE/*"
      }
   ]
}
```

3. Create a basic ReactJs Web App & Deploy it to the hosted S3 Bucket.

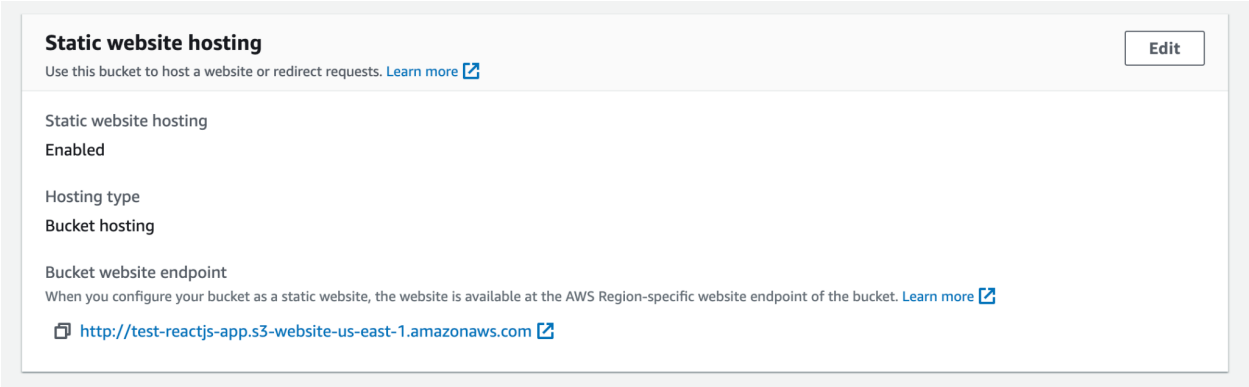
Since you have prepared the S3 bucket, you can start creating a very basic default provided ReactJs App by executing the below commands:

- $ node -v
- $ npm -v
- $ yarn create react-app serverless-test-app
- $ cd serverless-test-app
- $ yarn start (test the basic App on local)
- Now, to deploy the prepared & locally test ReactJs Web App to the hosted S3 Bucket, add deploy script in the package.json file.

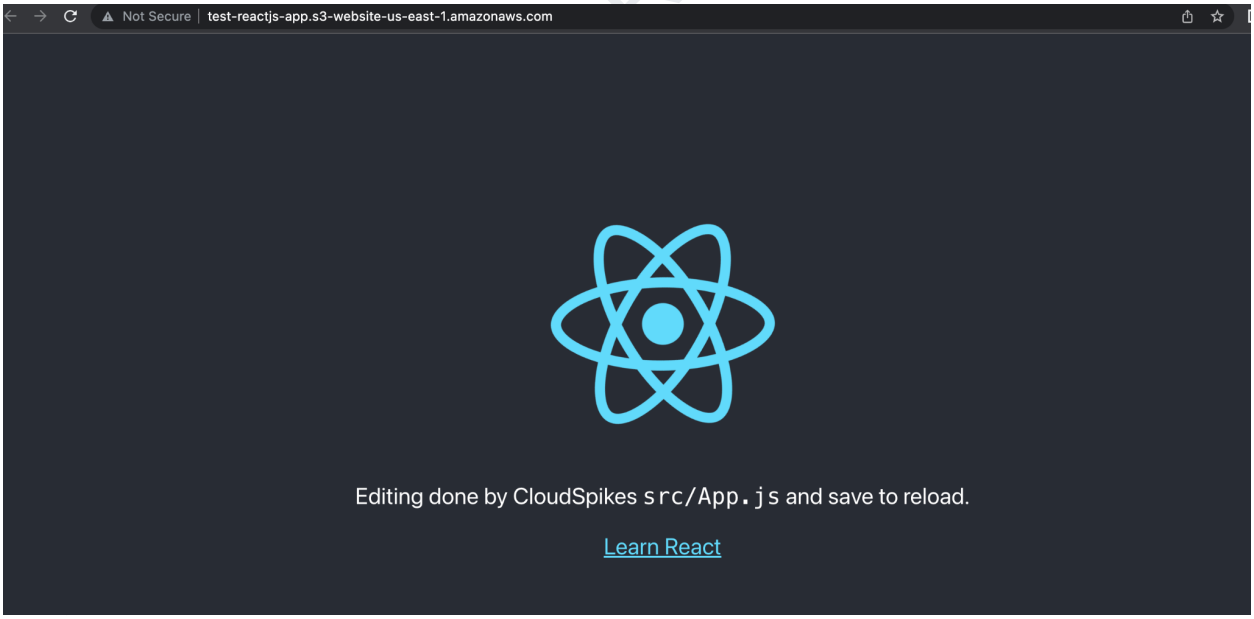
```
{
  "name": "serverless-test-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.14.1",
    "@testing-library/react": "^13.0.0",
    "@testing-library/user-event": "^13.2.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "deploy": "aws s3 sync build/ s3://test-reactjs-app"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
-- INSERT --
```

- $ Yarn build && yarn deploy

4. Finally you can test your deployed ReactJs Web App from the below AWS S3 Bucket Property tab:



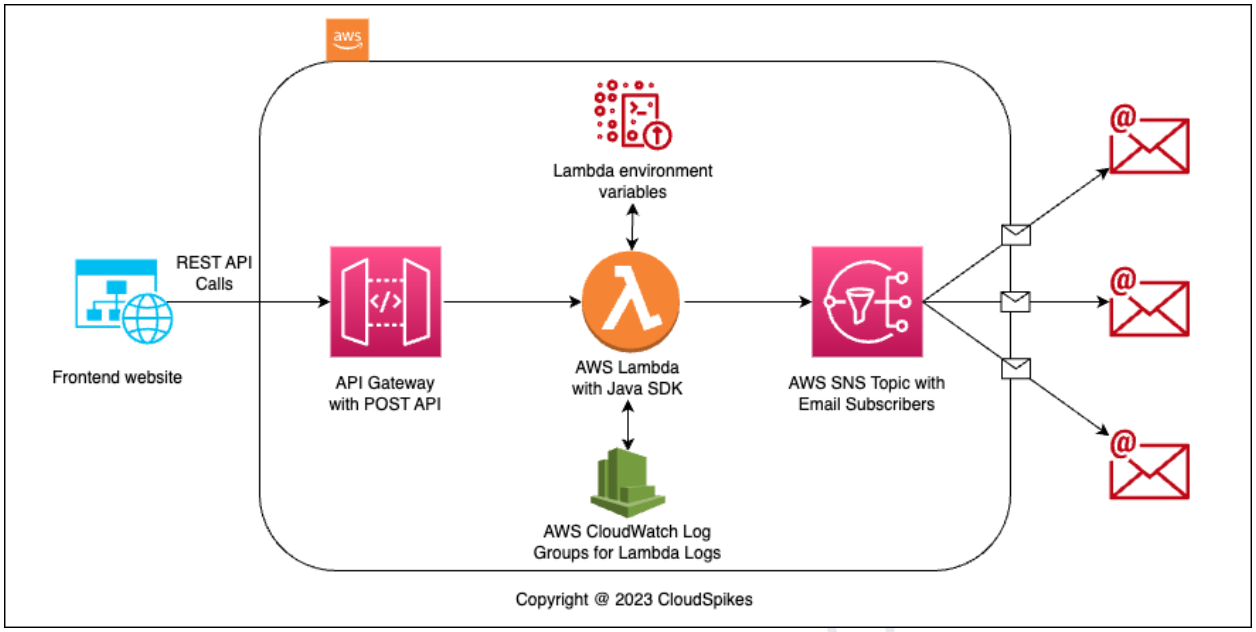
Hit the Bucket website endpoint link and you can see the deployed ReactJs Web App with the same view as it was found on your local environment while testing.



#9 Task: Prepare a Serverless Model Architecture that has Lambda with Java Code, API Gw integration, and SNS notifications using Email protocol.

1. **Here, the Lambda Java function will collect email adresses to send an email notification from the API Gw POST method.**
2. **Once, data is received, Lambda will publish email data to the SNS Topic and SNS will publish messages to the subscribed users via email protocol.**
3. **Configure IAM Roles for the AWS services as and where needed.**
4. **Use Lambda environment variables to inject an application credential (AK, SK, Region, Account ID, etc. as needed) to the running Java App on AWS Lambda Func.**

Monitor Logs from the CloudWatch Lambda-specific Log Group & Log Streams.



1. Java App Code - Create a Lambda Function with Java SDK.

- Open Eclipse and install AWS Toolkit to prepare AWS Lambda specific Java code using Maven build tool configurations using this AWS Official document.
- Now, from the menu bar, click on New → Project → Seach for "AWS Lambda" → Select **AWS Lambda Java Project** → Provide the name of the project and select the Input Type as **Stream Request Handler** → Hit Finish.
- This will create a new project in the project explorer section on left hand side with the project name folder you provided in the previous step.
- Now, refer to the source code from this GitHub Repository and prepare Java class files as well as pom.xml dependency file accordingly.
  Note: To skip the above steps, you can also simply import this GitHub Repository in to Eclipse IDE and compare the code with the GitHub code base to verify the Java and pom.xml files.
- FYI, this Java code will capture the input JSON data from the API Gateway POST API, process the input data as well as Create an SNS topic if does not exist and add subscribers. Also, this Java code will also publish the message received from the API Gateway POST API call on the configured SNS topic.
- Finally, to prepare the executable JAR build for this project, go to the terminal and navigate to the location of the AWS Lambda Java Project. Then. execute "**mvn clean install**" command. This will prepare the JAR file under **target** directory which will be uploaded to the AWS Lambda Function while deployment process will be executed. ***Note***: You can download the Java source code for this Lab work from this public GitHub Repo.

1. Infra Setup - Create a Lambda Function with Java SDK.

- Create a Lambda function and provide the basic details like Function name, Runtime as Java 11 (Corretto), Architecture type as x86_64 and Change default execution role option as Create a new role with basic Lambda permissions.
- Once, the Lambda is created with the basic details, go to Runtime settings and set Handler value as **com.amazonaws.lambda.demo.LambdaFunctionHandler::handleRequest**
- Here, **com.amazonaws.lambda.demo** is the Package value for the Java App, **LambdaFunctionHandler** is the Java Class name where Handler Function (App Java Code Entry Point from where Lambda execution code will start reading Java code from Line 0 to Line N - Last line of code) is defined and to define the Lambda Handler Function is distingueshed using :: (Double column). So, here, **handleRequest** is the Java Lambda Handler Function name.
- Furthermore, configure the environment variables needed for the Lambda function such as AWS_AK, AWS_SK, AWS_DEFAULT_REGION_VALUE and AWS_ACCOUNT_ID by navigating to **Configuration tab** → **Environment Variables** section.
- Deploy the JAR build of the Java App on the prepared AWS Lambda function by navigating to **Code tab** → **Code Source** section and selecting **Upload from** → **.zip or .jar file** and upload the built JAR file using Maven build tool from the build server/machine. In this case, we prepared JAR file on local env, so upload it from the local machine.

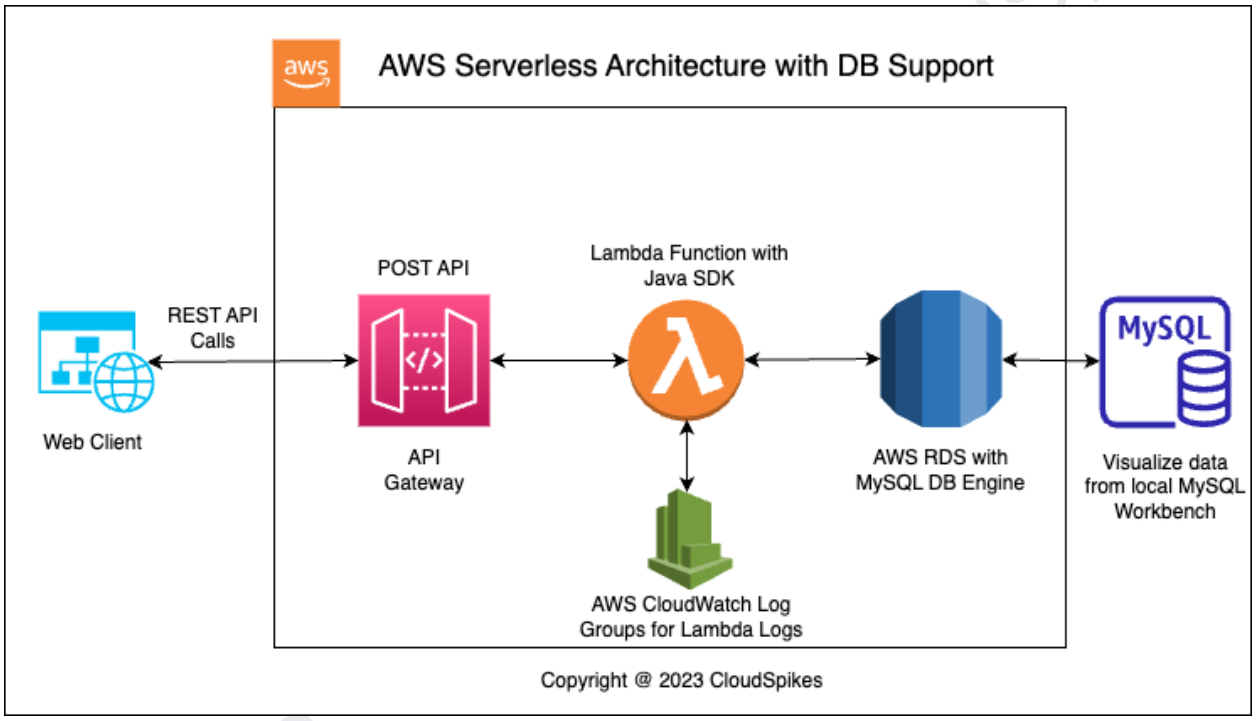
2. Infra Setup - Create a REST API with required Resource and Method.

- Create a REST API by going to the API Gateway AWS service.
- Define an API Resource like **test-send-email-notification** by clicking on Actions button and selecting the root / Resource.
- Once the Resource is defined, click on the created **test-send-email-notification** Resource and click on Create Method button via Actions button.
- Select a POST method for this API resource and then select integration type Lambda and map it with the created Lambda function in step 1.
- Finally, select the root / Resource and click on Actions button. Select Deploy API option and create/use the Stage to deploy the APIs defined.

3. Observe Email notifications on the subscribed users as well CloudWatch Logs from the Lambda specific Log Group and Log Stream.

- If there is a new Subcriber email address sent from the POST REST API, that email needs to be confirmed from the email inbox via email sent by Amazon SNS Service.
- Once, the subscribers are confirmed, they can start receiving email notification upon the POST API calls via a frontend website or a web API testing client like Postman tool.
- Also. Lambda execution logs can be observed by navigating to the respective Lambda function → Monitor tab → View CloudWatch Logs. This will open up a CloudWatch Log Group specific to the Lambda function you selected. Now, open up the Log Stream which you want to observe for the Java App Logs on Cloud Environment. FYI, you can go through the required App Logs by referring to the Log Stream's **Log event time** stamp value by comparing it with the time stamp when the API was called.

#10 Task: Create a POST API Gw API with Lambda that has a Java function to store data directly to the RDS DB Instance with MySQL DB Engine with the latest stable version. Visualize the data stored in the DB using MySQL Workbench or a similar tool.



1. Java App Code - Create a Lambda Function with Java SDK.

- Open Eclipse and install AWS Toolkit to prepare AWS Lambda specific Java code using Maven build tool configurations using this AWS Official document.
- Now, from the menu bar, click on New → Project → Seach for “AWS Lambda” → Select **AWS Lambda Java Project** → Provide the name of the project and select the Input Type as **Stream Request Handler** → Hit Finish.
- This will create a new project in the project explorer section on left hand side with the project name folder you provided in the previous step.
- Now, refer to the source code from this GitHub Repository and prepare Java class files as well as pom.xml dependency file accordingly.

Note: To skip the above steps, you can also simply import this GitHub Repository in to Eclipse IDE and compare the code with the GitHub code base to verify the Java and pom.xml files.

- FYI, this Java code will capture the input JSON data from the API Gateway POST API, process the input data as well as execute the SQL Queries on RDS MySQL Instance.
- Finally, to prepare the executable JAR build for this project, go to the terminal and navigate to the location of the AWS Lambda Java Project. Then. execute "**mvn clean install**" command. This will prepare the JAR file under **target** directory which will be uploaded to the AWS Lambda Function while deployment process will be executed. ***Note***: You can download the Java source code for this Lab work from [this public GitHub Repo](#).

1. Infra Setup - Setup RDS DB Service on AWS for MySQL DB Server.

- Create a RDS DB Instance by navigating to RDS service and clicking on **Create database** button.
- A web form will appear on the screen where you need to fill up all the configurable value to create a RDS Instance.
- Select a database creation method as **Standard create** and **DB Engine** type as MySQL (Not Aurora (MySQL Compatible) option).
- Leave rest of the values such as DB Version as latest with default values and choose template as **Free tier**.
- In the settings tab, leave DB Identifier as default value and enter **DB User** name and **Password** (Do not select Manage master credentials in AWS Secrets Manager and Auto generate a password check boxes).
- In the instance type selection, choose **db.t2.micro** to save cost in the practice lab work.
- Leave **Storage** configs as default values and in **Connectivity** select **Public access Yes** and select **Availability Zone** as per your preference.
- In the **Additional configuration** under **Database options** - set **Initial database name** as **rds_init_db**.
- Keep rest of the settings as default values and hit **Create database** button to finish creating the MySQL DB Instance.

Post RDS DB Creation Steps:

- Once the RDS DB Instance is up and running, you can navigate to the RDS Instance → **Connectivity & security** tab under **Endpoint & port** copy the Endpoint value.
- Go to your local MySQL workbench tool and test the created DB Instance by clicking on Database option from tool bar → Connect to Database…
- Paste endpoint value in the Hostname text box and DB user credentials as per the values provided while creating the RDS DB Instance.
- You can execute SQL queries in the MySQL Workbench Query editor tool and test the RDS DB Connectivity by executing a test SQL query as below:
  SELECT * FROM sys.sys_config;
- Once DB Connectivity testing is done successfully, you need to replace these DB Configuration valuesin the MySQL Connection Java file in your Lambda source code.

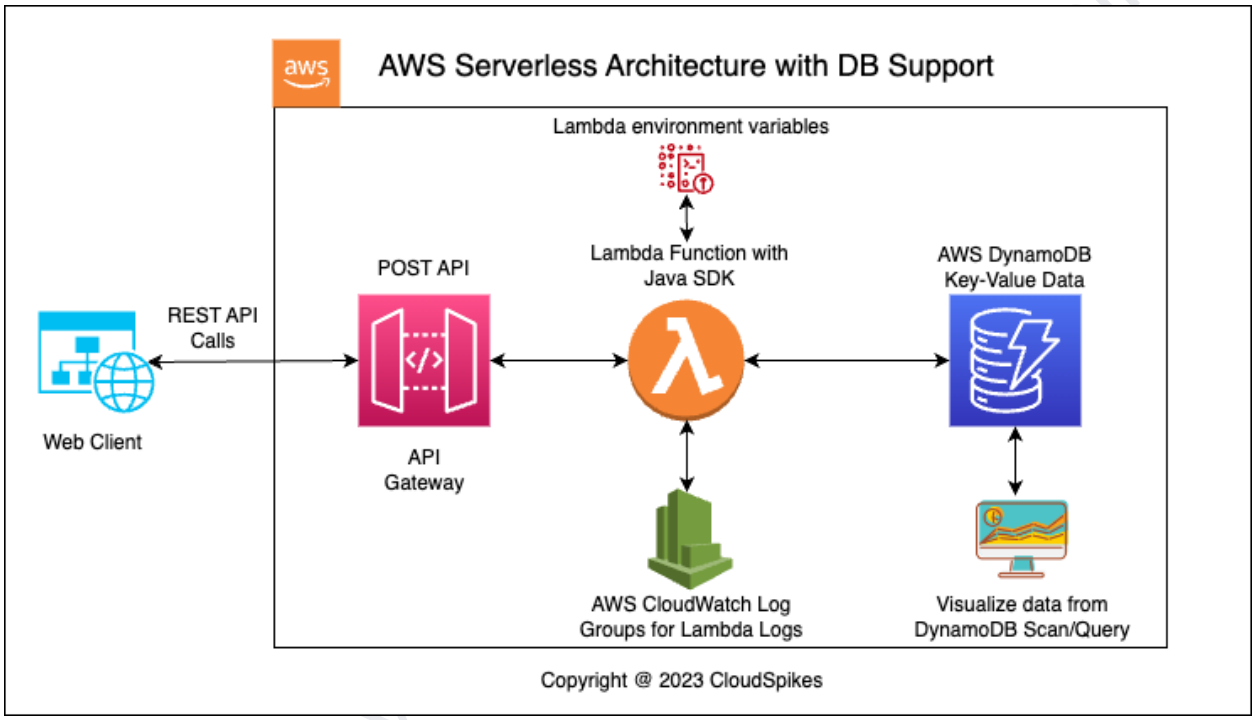
2. Infra Setup - Create a Lambda Function with Java SDK.

- Create a Lambda function and provide the basic details like Function name, Runtime as Java 11 (Corretto), Architecture type as x86_64 and Change default execution role option as Create a new role with basic Lambda permissions.
- Once, the Lambda is created with the basic details, go to Runtime settings and set Handler value as
  com.amazonaws.lambda.demo.LambdaFunctionHandler::handleRequest
- Here, **com.amazonaws.lambda.demo** is the Package value for the Java App, **LambdaFunctionHandler** is the Java Class name where Handler Function (App Java Code Entry Point from where Lambda execution code will start reading Java code from Line 0 to Line N - Last line of code) is defined and to define the Lambda Handler Function is distingueshed using :: (Double column). So, here, **handleRequest** is the Java Lambda Handler Function name.
- Furthermore, configure the environment variables needed for the Lambda function such as AWS_AK, AWS_SK, AWS_DEFAULT_REGION_VALUE and AWS_ACCOUNT_ID by navigating to **Configuration tab** → **Environment Variables** section.
- Deploy the JAR build of the Java App on the prepared AWS Lambda function by navigating to **Code tab** → **Code Source** section and selecting **Upload from** → **.zip or .jar file** and upload the built JAR file using Maven build tool from the build server/machine. In this case, we prepared JAR file on local env, so upload it from the local machine.

3. Infra Setup - Create a REST API with required Resource and Method.

- Create a REST API by going to the API Gateway AWS service.
- Define an API Resource like **test-rds-db-ops** by clicking on Actions button and selecting the root / Resource.
- Once the Resource is defined, click on the created **test-rds-db-ops** Resource and click on Create Method button via Actions button.
- Select a POST method for this API resource and then select integration type Lambda and map it with the created Lambda function in step 1.
- Finally, select the root / Resource and click on Actions button. Select Deploy API option and create/use the Stage to deploy the APIs defined.

#11 Task: Create a POST API Gw API with Lambda that has a Java function to store data directly to the DynamoDB Instance with the latest stable version. Visualize the data stored in the DB using DynamoDB Dashboard.



1. Java App Code - Create a Lambda Function with Java SDK.

- Open Eclipse and install AWS Toolkit to prepare AWS Lambda specific Java code using Maven build tool configurations using this AWS Official document.
- Now, from the menu bar, click on New → Project → Seach for “AWS Lambda” → Select **AWS Lambda Java Project** → Provide the name of the project and select the Input Type as **Stream Request Handler** → Hit Finish.
- This will create a new project in the project explorer section on left hand side with the project name folder you provided in the previous step.
- Now, refer to the source code from this GitHub Repository and prepare Java class files as well as pom.xml dependency file accordingly.

Note: To skip the above steps, you can also simply import this GitHub Repository in to Eclipse IDE and compare the code with the GitHub code base to verify the Java and pom.xml files.

- FYI, this Java code will capture the input JSON data from the API Gateway POST API, process the input data as well as Create DynamoDB Table if does not exist and add data inputs received from API Gateway POST API call.
- Finally, to prepare the executable JAR build for this project, go to the terminal and navigate to the location of the AWS Lambda Java Project. Then. execute "**mvn clean install**" command. This will prepare the JAR file under **target** directory which will be uploaded to the AWS Lambda Function while deployment process will be executed. ***Note***: You can download the Java source code for this Lab work from this public GitHub Repo.

1. Infra Setup - Create a Lambda Function with Java SDK.

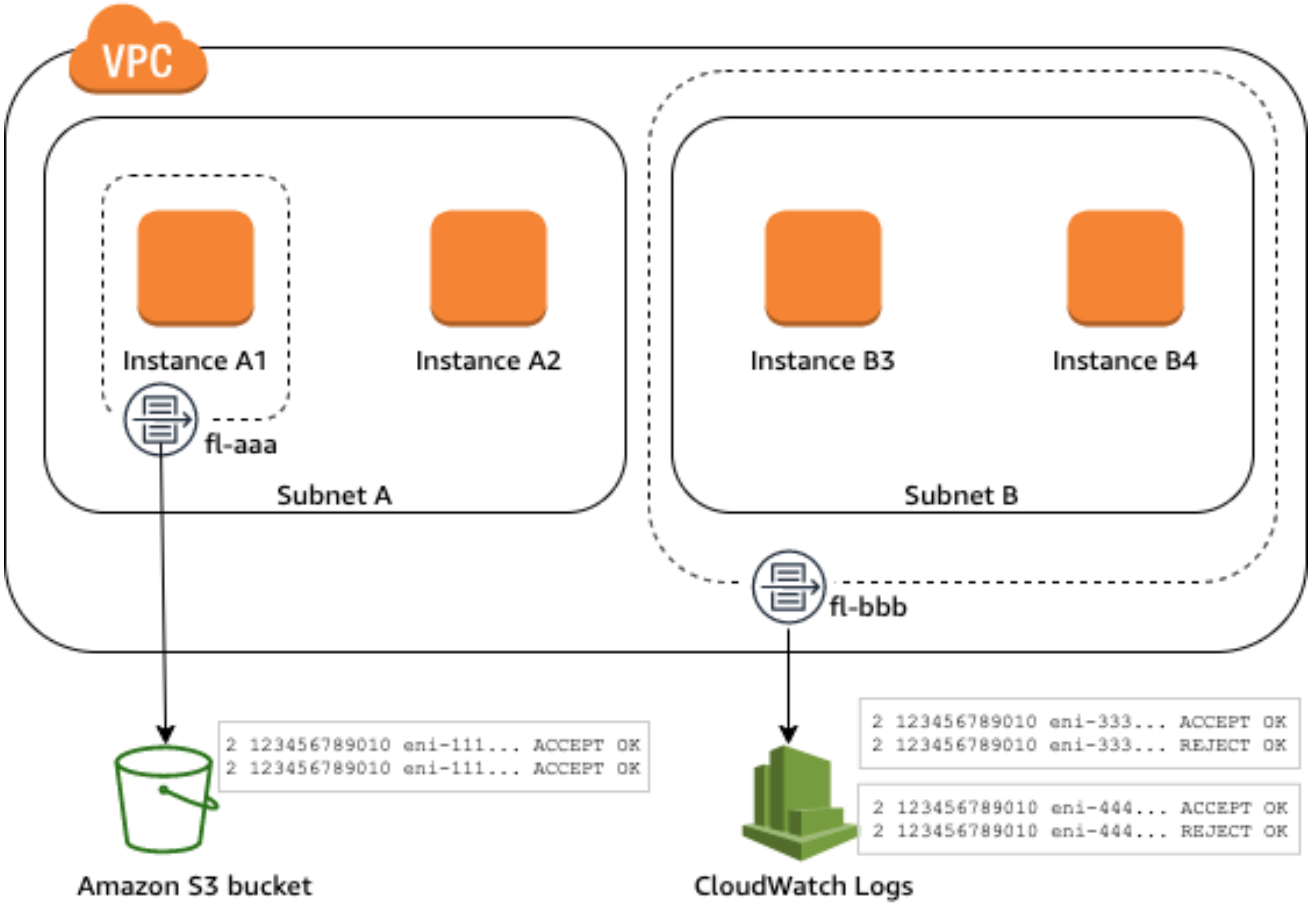
- Create a Lambda function and provide the basic details like Function name, Runtime as Java 11 (Corretto), Architecture type as x86_64 and Change default execution role option as Create a new role with basic Lambda permissions.
- Once, the Lambda is created with the basic details, go to Runtime settings and set Handler value as **com.amazonaws.lambda.demo.LambdaFunctionHandler::handleRequest**
- Here, **com.amazonaws.lambda.demo** is the Package value for the Java App, **LambdaFunctionHandler** is the Java Class name where Handler Function (App Java Code Entry Point from where Lambda execution code will start reading Java code from Line 0 to Line N - Last line of code) is defined and to define the Lambda Handler Function is distingueshed using :: (Double column). So, here, **handleRequest** is the Java Lambda Handler Function name.
- Furthermore, configure the environment variables needed for the Lambda function such as AWS_AK and AWS_SK by navigating to **Configuration tab** → **Environment Variables** section.
- Deploy the JAR build of the Java App on the prepared AWS Lambda function by navigating to **Code tab** → **Code Source** section and selecting **Upload from** → **.zip or .jar file** and upload the built JAR file using Maven build tool from the build server/machine. In this case, we prepared JAR file on local env, so upload it from the local machine.

2. Infra Setup - Create a REST API with required Resource and Method.

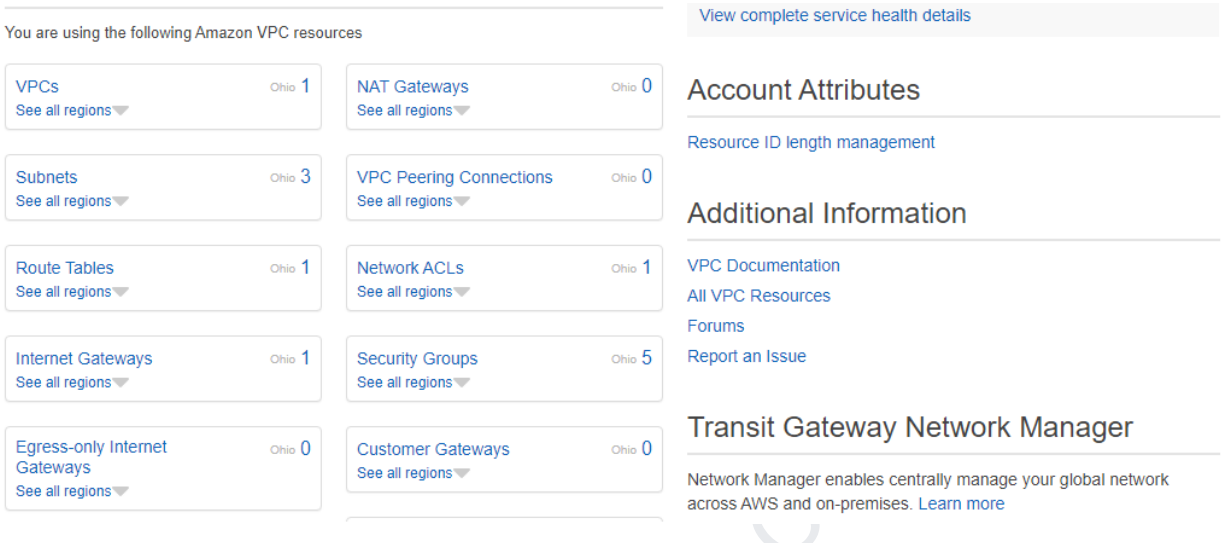
- Create a REST API by going to the API Gateway AWS service.
- Define an API Resource like **test-dyanmodb-ops** by clicking on Actions button and selecting the root / Resource.
- Once the Resource is defined, click on the created **test-dyanmodb-ops** Resource and click on Create Method button via Actions button.
- Select a POST method for this API resource and then select integration type Lambda and map it with the created Lambda function in step 1.
- Finally, select the root / Resource and click on Actions button. Select Deploy API option and create/use the Stage to deploy the APIs defined.
- You can test this deployed Stage Invoke API via a web client tool such as Postman.

#12 Task: **Create VPC Flow Logs, S3 Access Logs, API Gw Logs, Lambda Logs, CloudTrail logs and Install CloudWatch Monitoring Agents on EC2 to capture data for logs. All these Logs can be used further to configure a Cloud Vulnerability Scanning System to maintain a robust & secured Infra Environment.**

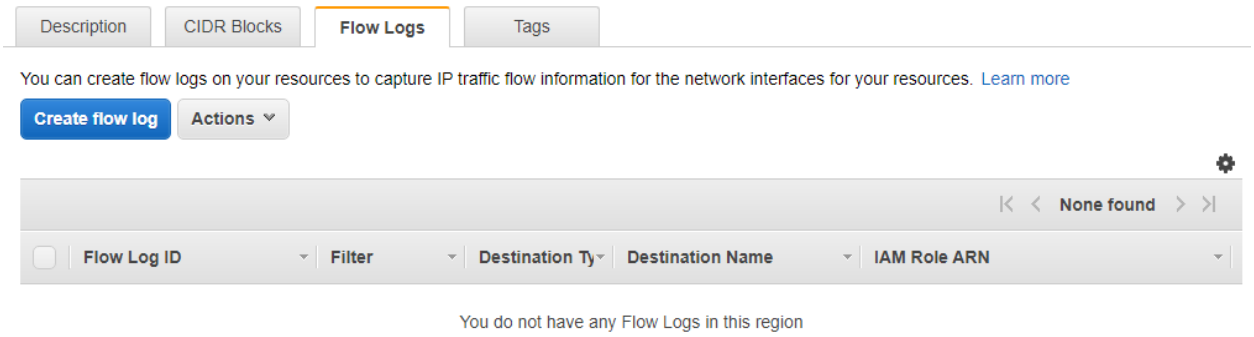
1. Setup VPC Flow Logs To CloudWatch Log Group Or S3 Bucket.



1. Go to your VPC Dashboard on the AWS console

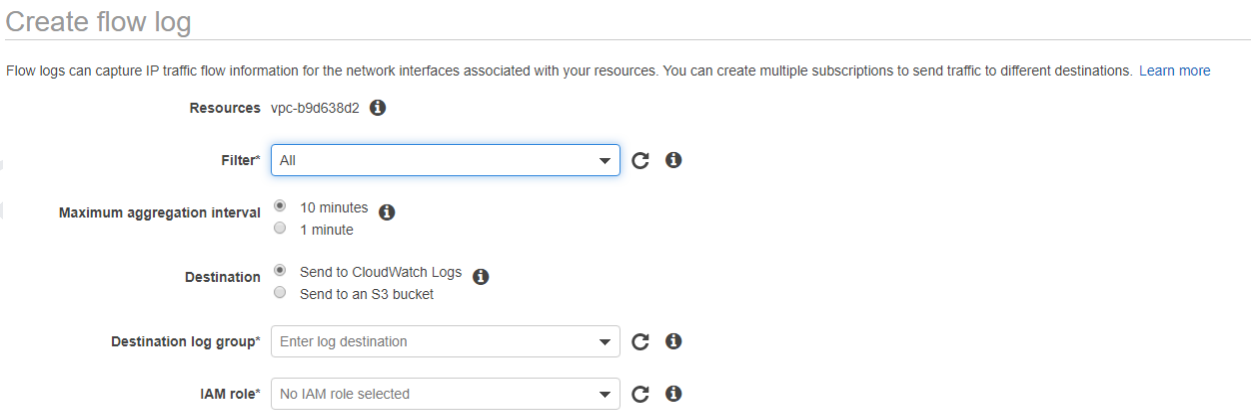


2. Click on Flow logs



Select Flow logs in the VPC Dashboard

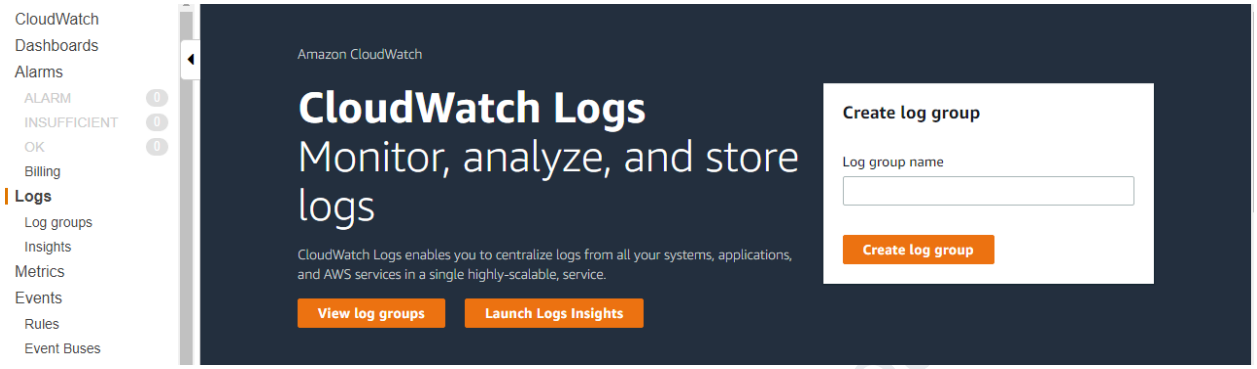
3. Click on Create Flow logs



Create Flow log

If you don't have any Destination log group or IAM Role create one from scratch.

- To Create a Destination log group, Go to the Cloudwatch dashboard on your VPC Console

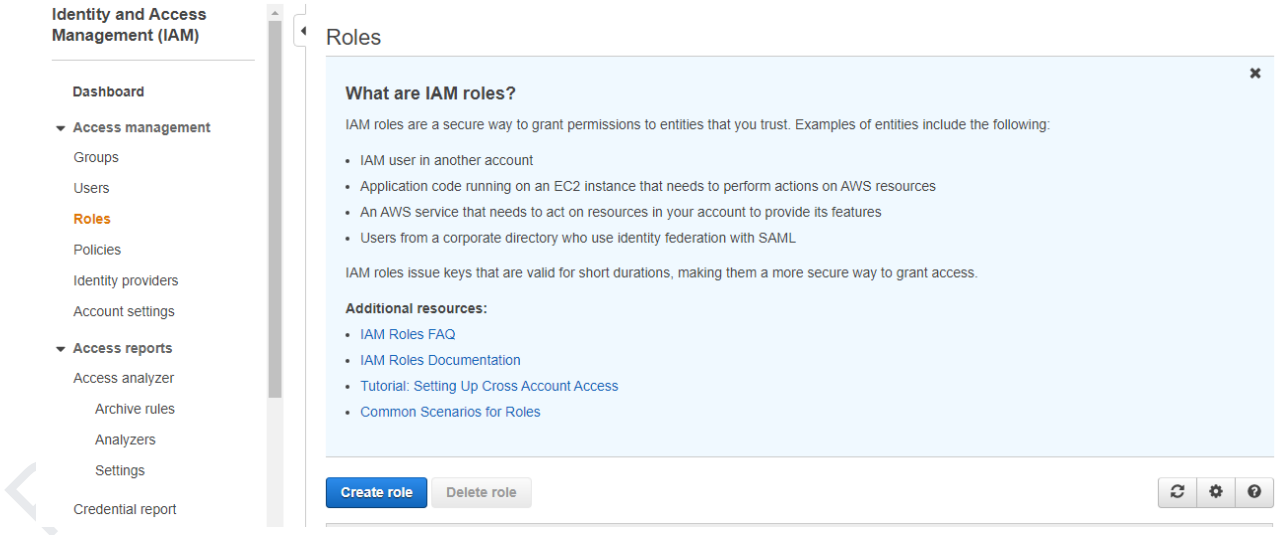


Click on Logs and enter a log group name, click on create log

4. Return back to the VPC Flowlogs dashboard and enter the Log group name created.

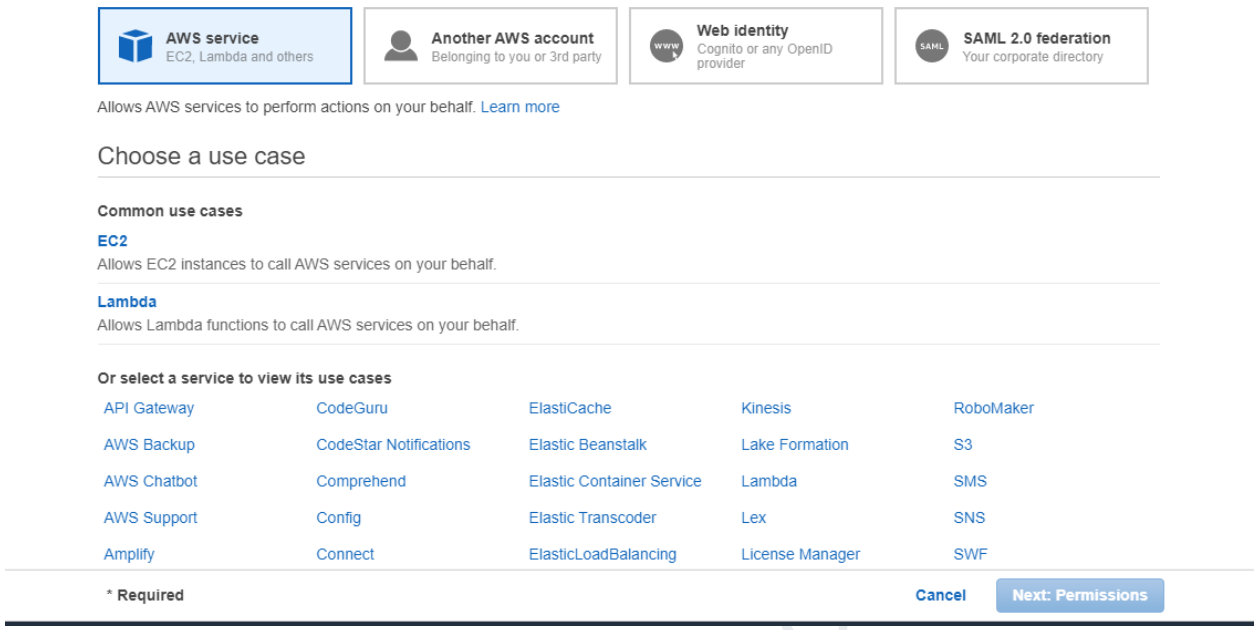
5. Create an IAM role which will allow VPC to Write to the log group

- To create an IAM role, go to the Identity and Access Management dashboard.



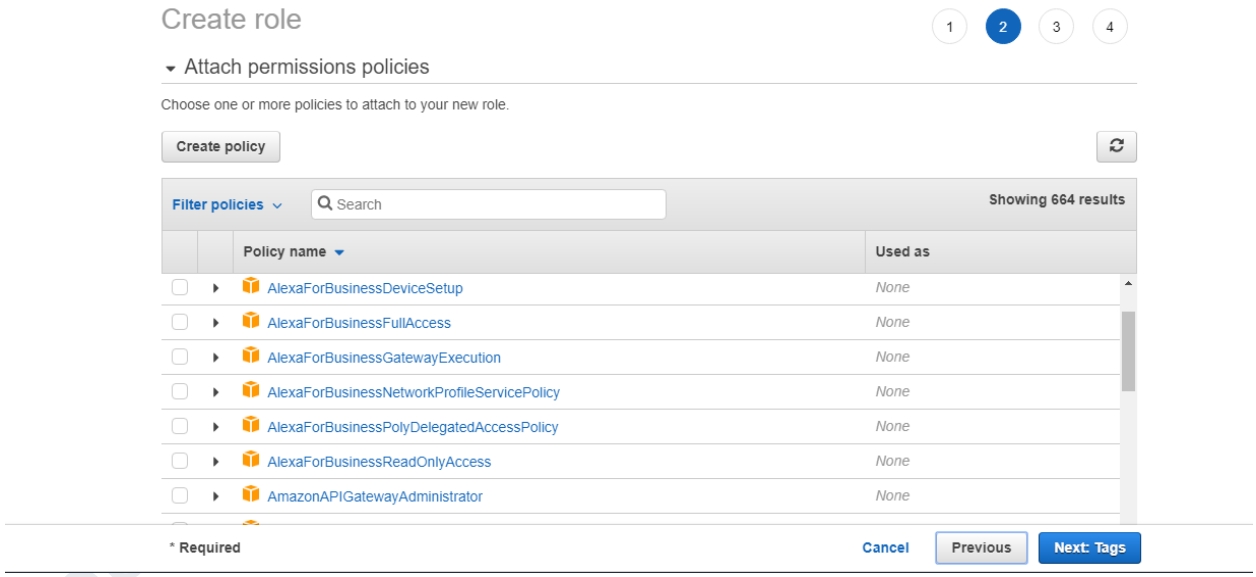
Click on Create a Role.

- The role that will be created will be an EC2 Role.



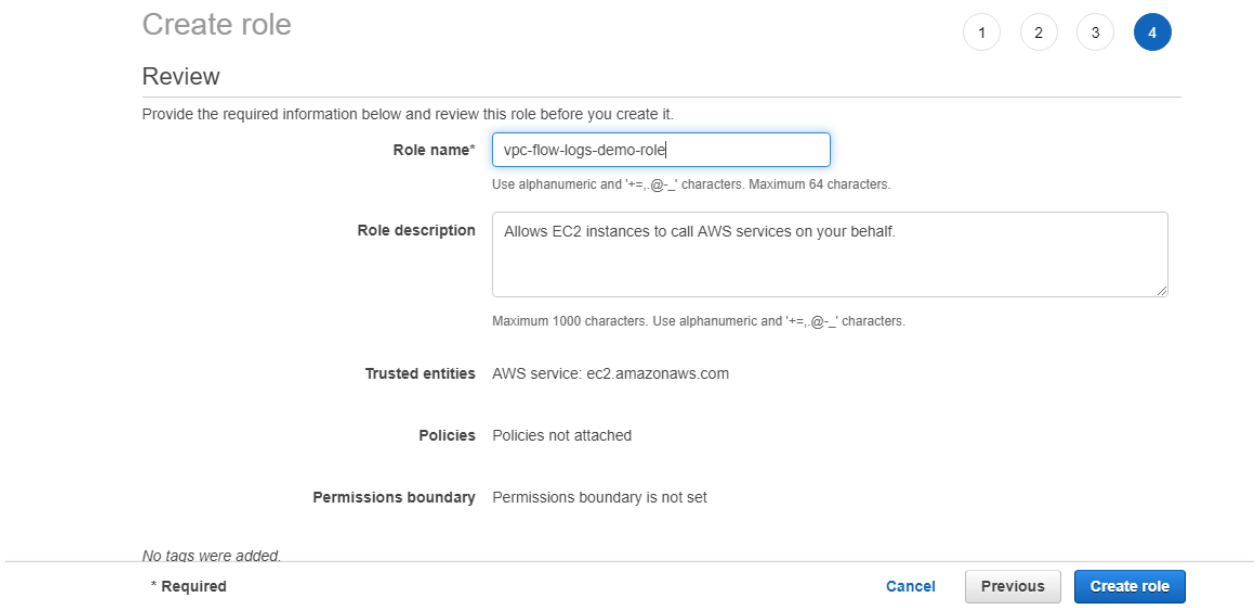
Select EC2 Role.

- Click Next to enter Tag names



Click on Next:Tags

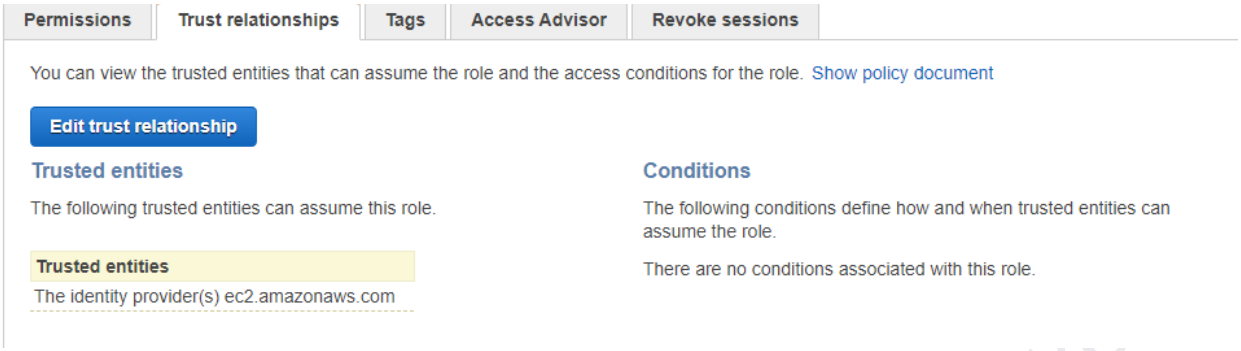
- Enter the Role name and click on create a role



- Attach the IAM roles for publishing flow logs to CloudWatch Logs
- Go to the Role you created and attach the inline policy

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "logs:CreateLogGroup",
    "logs:CreateLogStream",
    "logs:PutLogEvents",
    "logs:DescribeLogGroups",
    "logs:DescribeLogStreams"
   ],
   "Effect": "Allow",
   "Resource": "*"
  }
 ]
}
```

- Ensure that your role has a trust relationship that allows the flow logs service to assume the role. This allows EC2 to write into the Log-group.
- Click on Trust Relationship in the Roles Dashboard

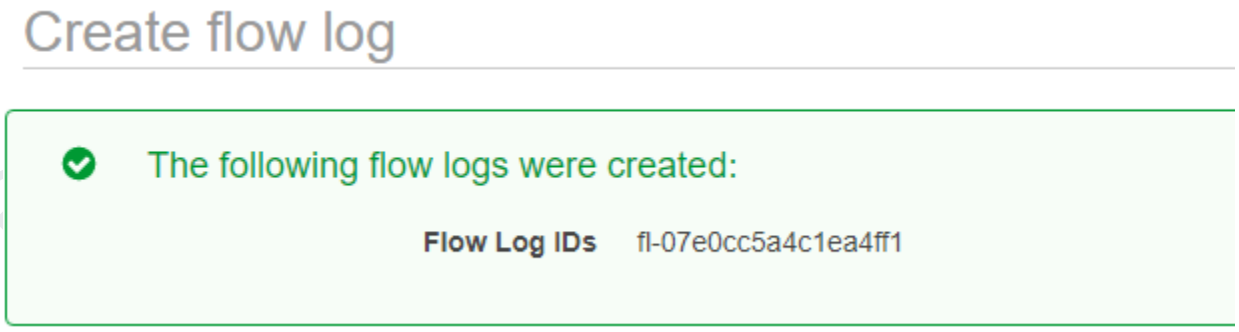


Click on Edit trust relationship

- Paste this there

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": {
    "Service": "vpc-flow-logs.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

6. Go back to your VPC Flow log dashboard and select the role you created. Click on Create flow logs.



The output of the flow log created

7. Wait for some time, return back to the CloudWatch dashboard and click on log groups you will see the Log stream Traffic.

```
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 80 40642 6 80 52700 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 18.185.97.81 172.31.35.55 40096 80 6 80 5800 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 18.185.97.81 172.31.35.55 40398 80 6 79 5748 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 80 41668 6 81 52752 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 18.185.97.81 172.31.35.55 80 40604 6 82 52804 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 41970 80 6 77 5644 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 18.185.97.81 172.31.35.55 80 41930 6 80 52700 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 80 41094 6 80 52700 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 42022 80 6 80 5800 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 41212 80 6 80 5800 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 80 40466 6 79 52648 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 80 41182 6 81 52752 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 80 39788 6 78 52596 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 40188 80 6 80 5800 1534192449 1534192626 ACCEPT OK
2 579807160478 eni-07ec21a29debdea6a 172.31.35.55 18.185.97.81 39204 80 6 79 5748 1534192449 1534192626 ACCEPT OK
```

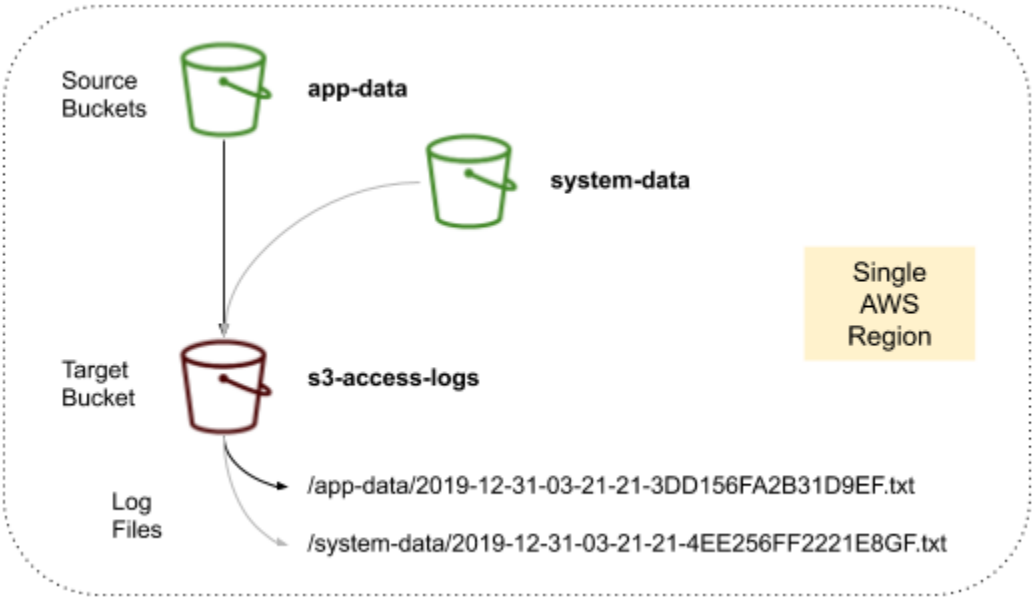
How a typical Flow log looks like

In the Flow log dashboard you can also filter traffic coming from a specific IP address.

Big Ups! You were able to capture information about the IP traffic in your VPC, storing the raw data in Amazon CloudWatch where it can be retrieved and viewed.

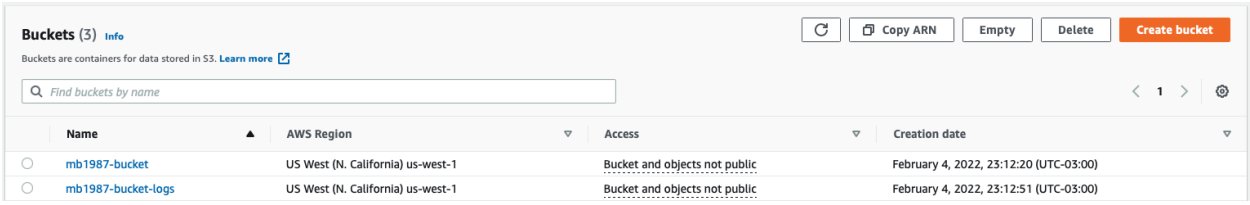
Here is the ***official AWS documentation*** for the same process we gone through above.

2. Setup S3 Access Logs to Log Management S3 Bucket.

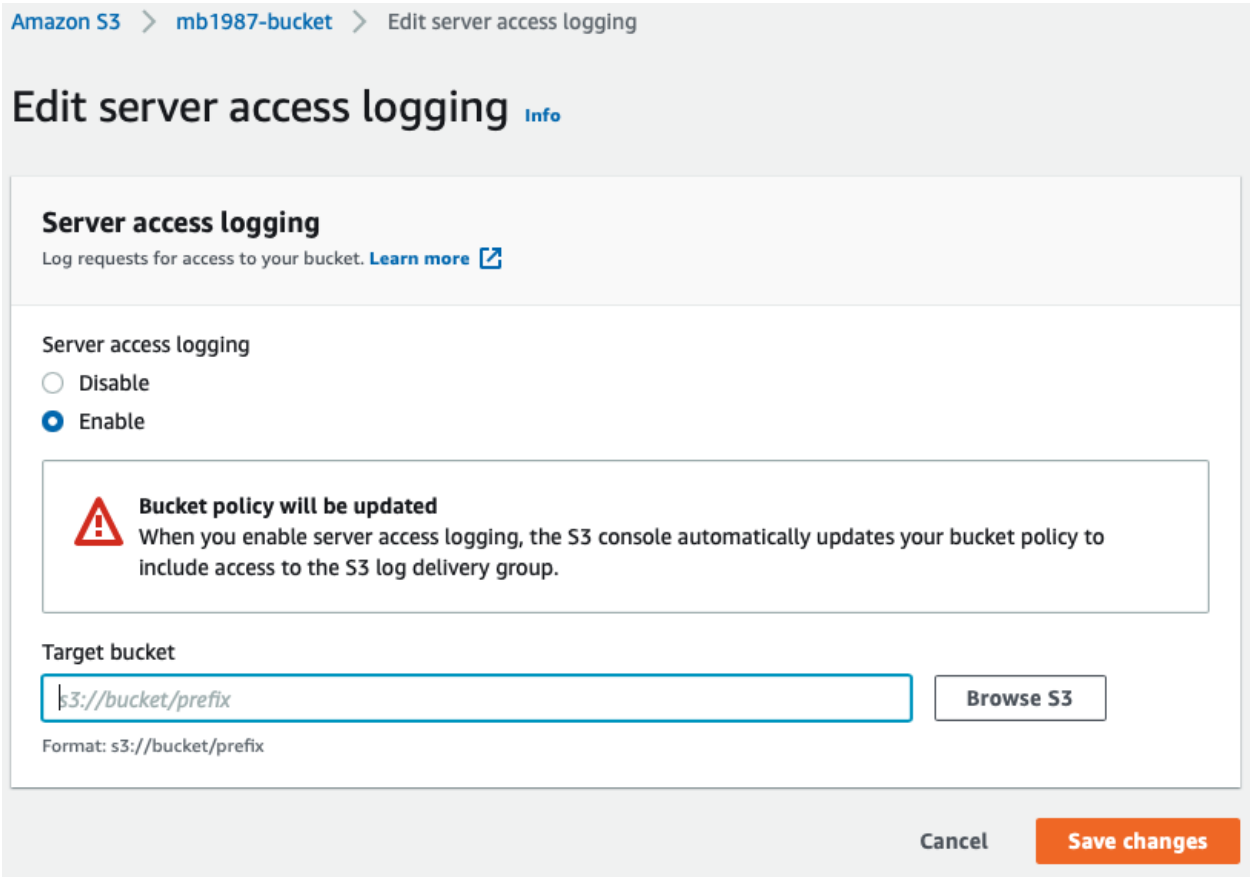


1. Enabling Logging for Bucket Objects

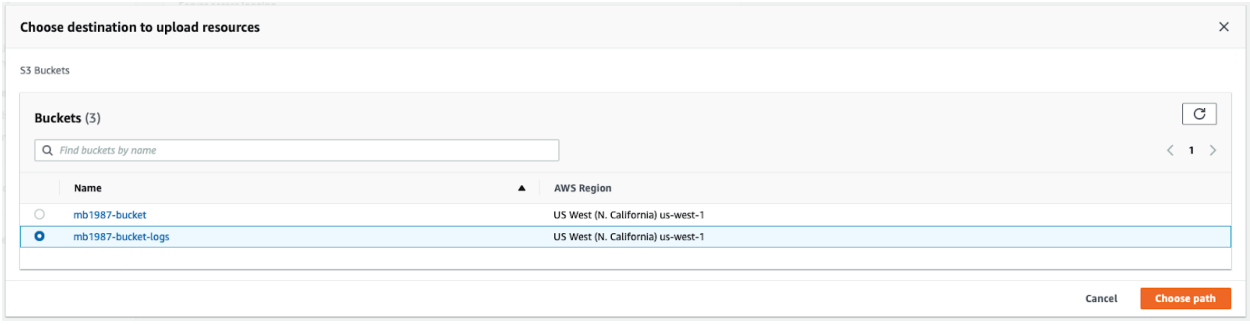
To use S3 logs, you first need to create one bucket to store files (objects) and another to store the logs. This should be created in the same region. It is a good practice not to save the logs in the same bucket because we want to save the logs for the interactions that the bucket receives and if the bucket has a problem the logs may not be able to be saved with the information about what is causing the error.



After you've created the buckets, go to the Properties of the bucket that will store the files to associate it with the bucket for logs. On the Properties page, click on the Edit button in the Server access logging box. In this form, select Enable to allow the bucket to provide log data about stored objects, then click on Browse S3 to select the log bucket.

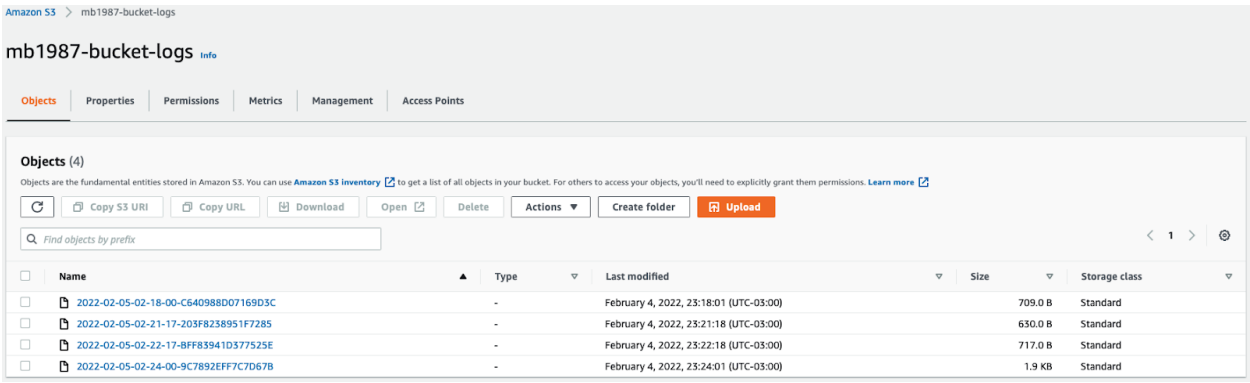


In the modal, select the proper bucket and click on Choose path. Back in the form, click on Save changes to apply the association between the buckets. Clicking that button is all you need to do to start saving object usage logs.



2. Testing Logging

Now, let's try to access the bucket and upload some files. For this test, there is no need to add any other settings. You can open, download, and remove files to generate logs for these actions. Then, you can access the bucket for logs and wait a few minutes to receive the log data about the newly uploaded file. Then, open the logs to see the type of data available in the log information. You will see logs related to actions taken within the bucket to get or remove objects, along with policies and versioning information.



3. Log Data Samples

Let's take a look at some sample logs and their available formats on AWS S3. For example, this is DELETE:

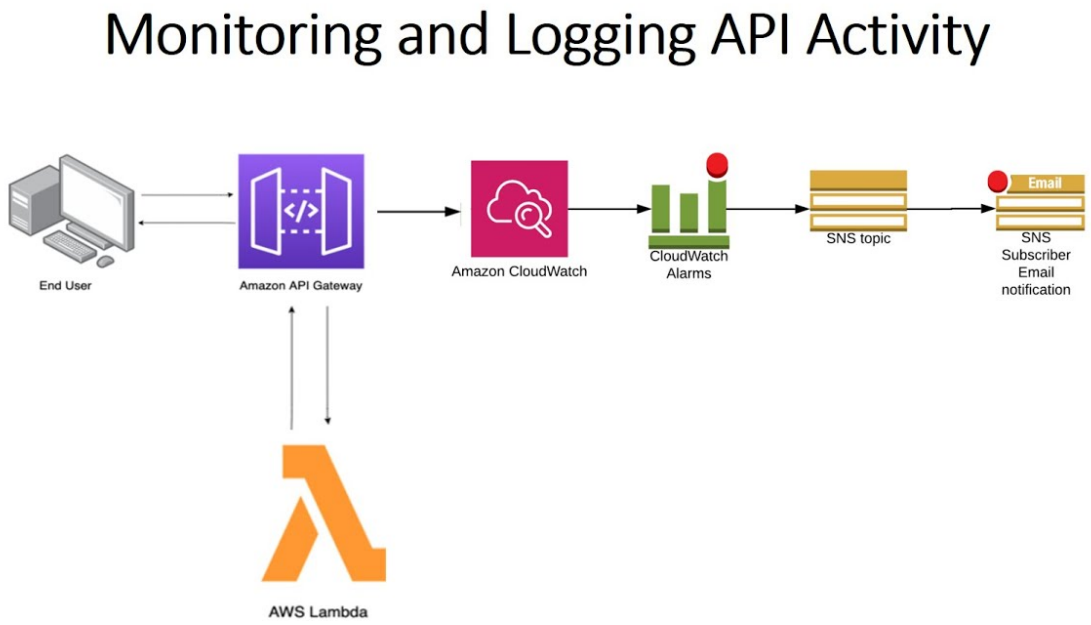
```
ce6f2c543de2b9a3a4fdf21d56e95135af4045032a9157cb2fdb1a4854c73110 mb1987-bucket [05/Feb/2022:01:09:08 +0000]
191.XXX.XXX.216 ce6f2c543de2b9a3a4fdf21d56e95135af4045032a9157cb2fdb1a4854c73110 BW21GJ60HATK1VCR
REST.POST.MULTI_OBJECT_DELETE - "POST /mb1987-bucket?delete= HTTP/1.1" 200 - 5100 - 802 - "-" "S3Console/0.4,
aws-internal/3 aws-sdk-java/1.11.1030 Linux/5.4.172-100.336.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/25.302-
b08 java/1.8.0_302 vendor/Oracle_Corporation cfg/retry-mode/standard" -
NLZUuMfnLn6Kq3LKEB5GFRNAVEo9cy/BR5bmzhy4dXWD0ogwa6Q71IzUbJKlidFbVwUiRR9jk9w= SigV4 ECDHE-RSA-AES128-GCM-
SHA256 AuthHeader s3-us-west-1.amazonaws.com TLSv1.2 -
```

And this is GET:

```
ce6f2c543de2b9a3a4fdf21d56e95135af4045032a9157cb2fdb1a4854c73110 mb1987-bucket [05/Feb/2022:01:08:58 +0000]
191.XXX.XXX.216 ce6f2c543de2b9a3a4fdf21d56e95135af4045032a9157cb2fdb1a4854c73110 MSAHCXNXGA3NJY9V
REST.GET.BUCKET - "GET /mb1987-bucket?list-type=2&encoding-type=url&max-keys=1&fetch-
owner=true&delimiter=&prefix= HTTP/1.1" 200 - 768 - 106 105 "-" "S3Console/0.4, aws-internal/3 aws-sdk-
java/1.11.1030 Linux/5.4.172-100.336.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/25.302-b08 java/1.8.0_302
vendor/Oracle_Corporation cfg/retry-mode/standard" -
X4VZjjWLHbLLApCiTeXoulUTQmYYduxisCY8E6se4YZx7FRBX6KsSd4lG+VwfrpgLqQXTdPbTGE= SigV4 ECDHE-RSA-AES128-GCM-
SHA256 AuthHeader s3-us-west-1.amazonaws.com TLSv1.2 -
```

Here is the ***official AWS documentation*** for the same process we have gone through above.

3. CloudWatch API logging using the API Gateway console



To set up CloudWatch API logging, you must have deployed the API to a stage. You must also have configured an appropriate CloudWatch Logs role ARN for your account.

1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway
2. Choose a REST API.
3. Choose Settings from the primary navigation panel and enter an ARN of an IAM role with appropriate permissions in CloudWatch log role ARN. You need to do this once.

4. Do one of the following:
    a. Choose an existing API and then choose a stage.
    b. Create an API and deploy it to a stage.
5. Choose Logs/Tracing in the Stage Editor.
6. To enable execution logging:
    a. Choose a logging level from the CloudWatch Logs dropdown menu.

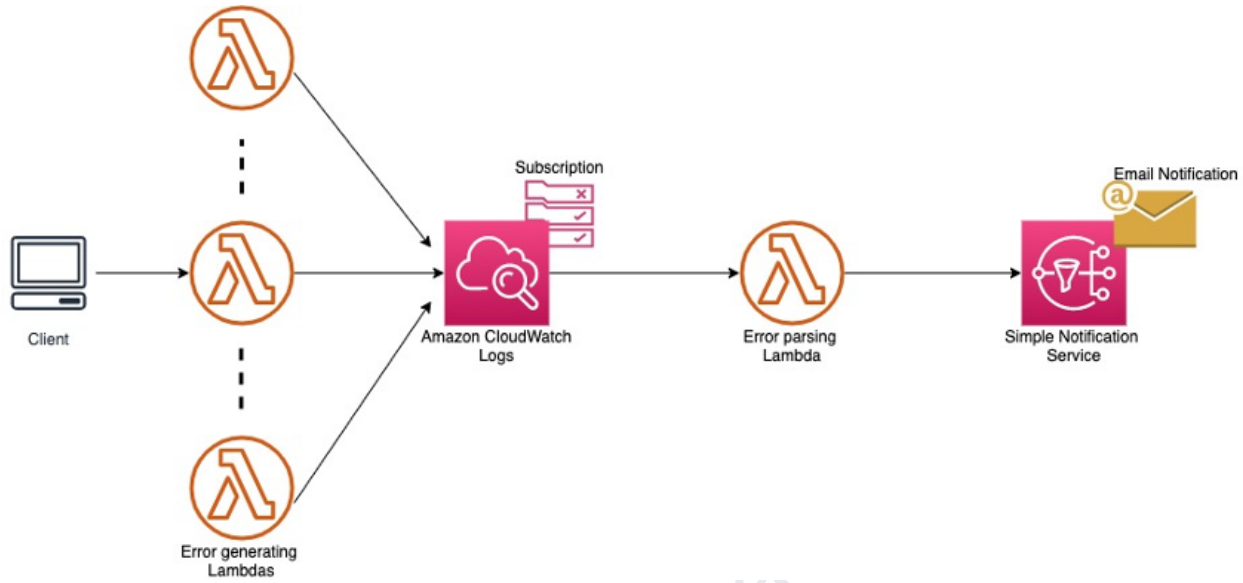
       Warning: *Full Request and Response Logs can be useful to troubleshoot APIs, but can result in logging sensitive data. We recommend not using Full Request and Response Logs for production APIs.*

    b. If desired, choose Enable Detailed CloudWatch Metrics.
7. For more information about CloudWatch metrics, see Monitoring REST API execution with Amazon CloudWatch metrics.
8. To enable access logging:
    a. Choose Enable Access Logging under Custom Access Logging.
    b. Enter the ARN of a log group in Access Log Destination ARN. The ARN format is arn:aws:logs:{region}:{account-id}:log-group:log-group-name.
    c. Enter a log format in Log Format. You can choose CLF, JSON, XML, or CSV to use one of the provided examples as a guide.
9. Choose to Save Changes.

Note: *You can enable execution logging and access logging independently of each other.*

API Gateway is now ready to log requests to your API. You don't need to redeploy the API when you update the stage settings, logs, or stage variables.

4. Monitoring AWS Lambda Function Logs in CloudWatch Stream specific to Lambda Function CloudWatch Log Group.



Prerequisites:

Execution role attached to the Lambda Function needs permission to upload logs to CloudWatch Logs. You can add CloudWatch Logs permissions using the AWSLambdaBasicExecutionRole AWS managed policy provided by Lambda.

To add this policy to your role, run the following AWS CLI command *(Make sure you have configure AWS CLI using Access Key / Secret Key on the machine where you are executing the below AWS CLI command)*:

aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

For more information, see AWS managed policies for Lambda features.

Pricing:

There is no additional charge for using Lambda logs; however, standard CloudWatch Logs charges apply. For more information, see CloudWatch pricing.

Using the Lambda console to view Lambda execution logs in CloudWatch:

To view logs using the Lambda console

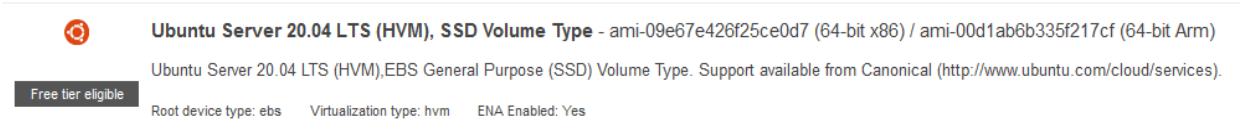
1. Open the Functions page of the Lambda console.

2. Choose a function.
3. Choose **Monitor.**
4. Choose **View logs in CloudWatch.**

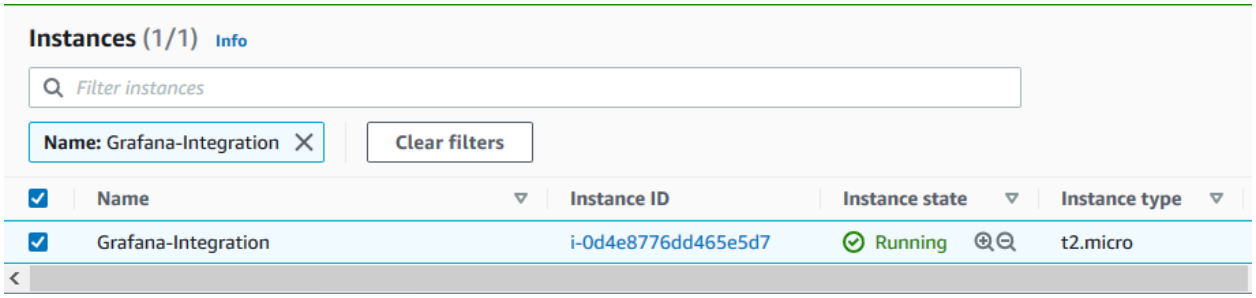
5. Capturing EC2 instance CPU/Mem/Disk Utilisation metrics by installing CW Agent on the server and monitoring them using a Custom CW Dashboard.

Step 1- Launch an EC2 with Ubuntu AMI 20.04

I believe that we can launch its own so I'm not explaining the process for it.
Used AMI: (Ubuntu 20.04)



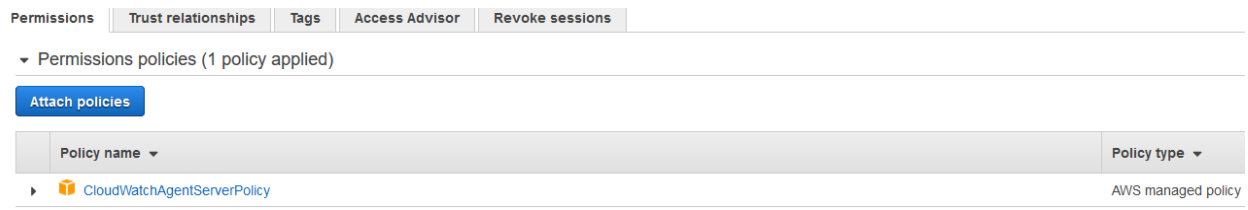
EC2 Instance: (Type: t2.micro)



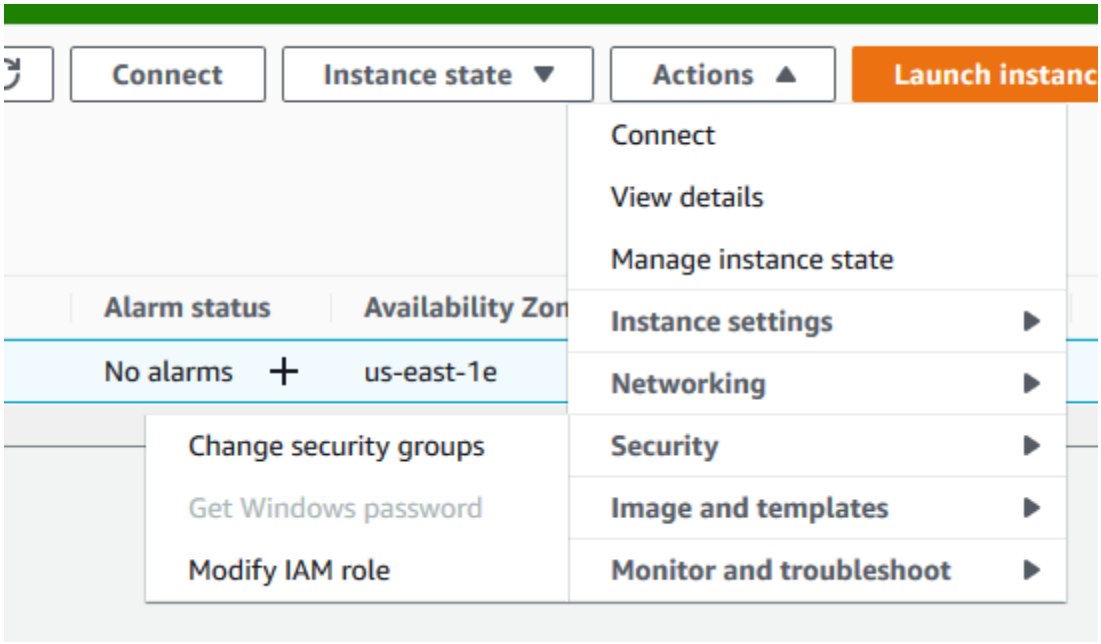
Step 2: Attach an IAM role with EC2 Instance

Access to AWS resources requires permissions. You create an IAM role, an IAM user, or both to grant permissions that the CloudWatch agent needs to write metrics to CloudWatch. If you're going to use the agent on Amazon EC2 instances, you must create an IAM role. If you're going to use the agent on on-premises servers, you must create an IAM user.

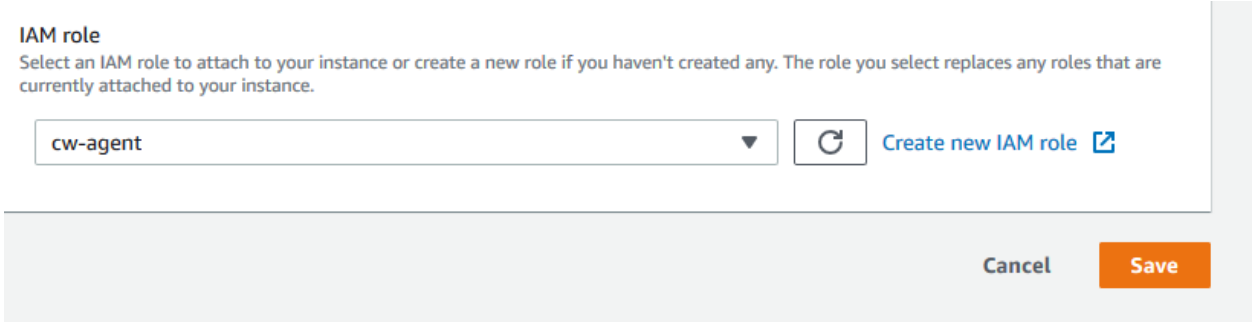
Create an IAM role, Make sure that AWS service is selected under Select type of trusted entity. For Choose a use case, choose EC2 under Common use cases, and Choose Next: Permissions. In the list of policies, select CloudWatchAgentServerPolicy & create an IAM role.



Now attach an IAM role with EC2 Instance. Go to EC2 — Select Instance — Click on Action — Security — Modify IAM role



Choose Create IAM role & save it.



Step 3: Download the CloudWatch Agent Package

Use the following steps to download the CloudWatch agent package, SSH to Instance & Download CW Agent package.

Download the CloudWatch agent:

wget https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb

```
root@ip-172-31-52-40:/home/ubuntu# wget https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb
--2021-05-04 05:34:37--  https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.1.27
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.1.27|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 54662644 (52M) [application/octet-stream]
Saving to: 'amazon-cloudwatch-agent.deb'

amazon-cloudwatch-agent.deb             100%[=======================================================================================>]  52.13M  47.0MB/s    in 1.1s

2021-05-04 05:34:38 (47.0 MB/s) - 'amazon-cloudwatch-agent.deb' saved [54662644/54662644]

root@ip-172-31-52-40:/home/ubuntu#
```

Install the package:

dpkg -i -E ./amazon-cloudwatch-agent.deb

```
root@ip-172-31-52-40:/home/ubuntu# dpkg -i -E ./amazon-cloudwatch-agent.deb
Selecting previously unselected package amazon-cloudwatch-agent.
(Reading database ... 60149 files and directories currently installed.)
Preparing to unpack ./amazon-cloudwatch-agent.deb ...
create group cwagent, result: 0
create user cwagent, result: 0
Unpacking amazon-cloudwatch-agent (1.247347.6b250880-1) ...
Setting up amazon-cloudwatch-agent (1.247347.6b250880-1) ...
root@ip-172-31-52-40:/home/ubuntu#
```

Update Packages & Install collectd: (This will take a few minutes if you haven't updated your available updates prior)

apt-get update && apt-get install collectd

```
0 upgraded, 557 newly installed, 0 to remove and 4 not upgraded.
Need to get 216 MB of archives.
After this operation, 1150 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 libsane-common all 1.0.29-0ubuntu5.2 [277 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 libtalloc2 amd64 2.3.0-3ubuntu1 [29.5 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 libtevent0 amd64 0.10.1-4 [35.5 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 libwbclient0 amd64 2:4.11.6+dfsg-0ubuntu1.8 [221 kB]
```

```
aspell-autobuildhash: processing: en [en_US-w_accents-only].
aspell-autobuildhash: processing: en [en_US-wo_accents-only].
Processing triggers for libgdk-pixbuf2.0-0:amd64 (2.40.0+dfsg-3ubuntu0.2) ...
Processing triggers for rygel (0.38.3-1ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Processing triggers for dbus (1.12.16-2ubuntu2.1) ...
Processing triggers for systemd (245.4-4ubuntu3.6) ...
Processing triggers for sgml-base (1.29.1) ...
root@ip-172-31-52-40:/home/ubuntu#
```

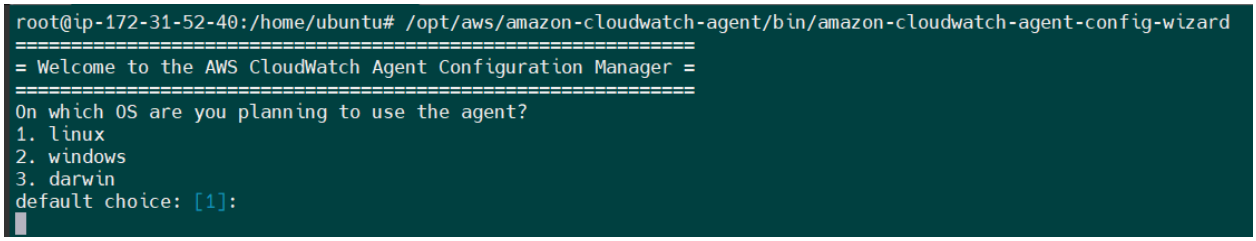
Step 4: Create the CloudWatch Agent Configuration File

Before running the CloudWatch agent on any servers, you must create a CloudWatch agent configuration file. The agent configuration file is a JSON file that specifies the

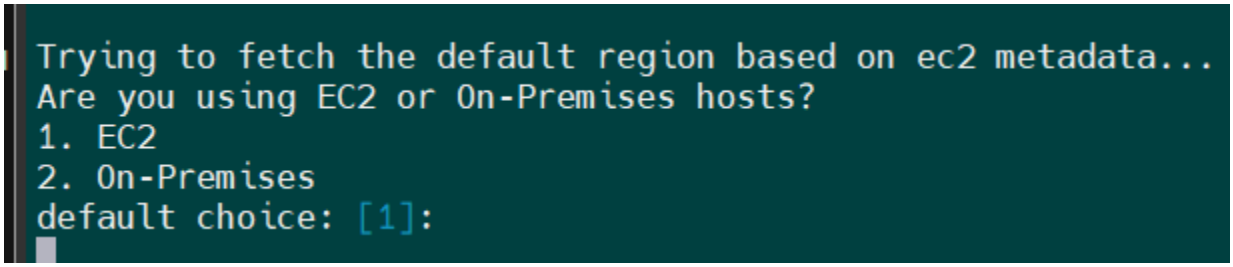
metrics and logs that the agent is to collect, including custom metrics. The agent configuration file wizard, amazon-cloud watch-agent-config-wizard, asks for a series of questions.

/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard

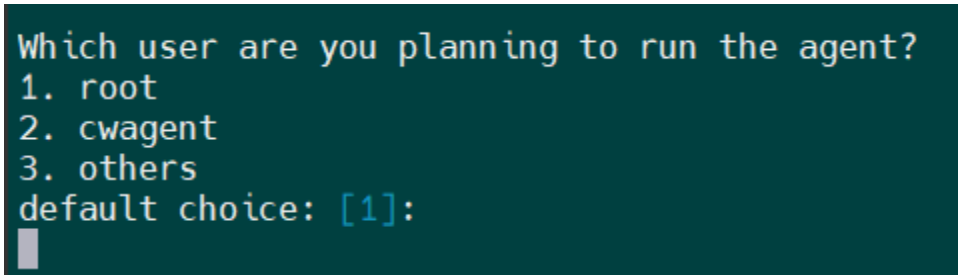
We're using Linux machine so option 1



Using EC2 Instance so Option 1



Select a user



We can skip these two options for memory metric.

```
Do you want to turn on StatsD daemon?
1. yes
2. no
default choice: [1]:
2
Do you want to monitor metrics from CollectD?
1. yes
2. no
default choice: [1]:
2
```

We can choose the below options according to our requirements.

```
Do you want to monitor any host metrics? e.g. CPU, memory, etc.
1. yes
2. no
default choice: [1]:
1
Do you want to monitor cpu metrics per core? Additional CloudWatch charges may apply.
1. yes
2. no
default choice: [1]:
2
Do you want to add ec2 dimensions (ImageId, InstanceId, InstanceType, AutoScalingGroupName) into all of your metrics if the info is available?
1. yes
2. no
default choice: [1]:
2
Would you like to collect your metrics at high resolution (sub-minute resolution)? This enables sub-minute resolution for all metrics, but you
cific metrics in the output json file.
1. 1s
2. 10s
3. 30s
4. 60s
default choice: [4]:
4
Which default metrics config do you want?
1. Basic
2. Standard
3. Advanced
4. None
default choice: [1]:
```

Provide some declarations for config files.

```
Are you satisfied with the above config? Note: it can be manually customized after the wizard completes to add additional items.
1. yes
2. no
default choice: [1]:
1
Do you have any existing CloudWatch Log Agent (http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AgentReference.html) confi
on?
1. yes
2. no
default choice: [2]:
2
Do you want to monitor any log files?
1. yes
2. no
default choice: [1]:
2
Saved config file to /opt/aws/amazon-cloudwatch-agent/bin/config.json successfully.
Current config as follows:
```

We can check json file under /opt/aws/amazon-cloudwatch-agent/bin/config.json

```
Saved config file to /opt/aws/amazon-cloudwatch-agent/bin/config.json successfully.
Current config as follows:
{
        "agent": {
                "metrics_collection_interval": 60,
                "run_as_user": "root"
        },
        "metrics": {
                "metrics_collected": {
                        "disk": {
                                "measurement": [
                                        "used_percent"
                                ],
                                "metrics_collection_interval": 60,
                                "resources": [
                                        "*"
                                ]
                        },
                        "mem": {
                                "measurement": [
                                        "mem_used_percent"
                                ],
                                "metrics_collection_interval": 60
                        }
                }
        }
}
Please check the above content of the config.
The config file is also located at /opt/aws/amazon-cloudwatch-agent/bin/config.json.
Edit it manually if needed.
Do you want to store the config in the SSM parameter store?
1. yes
2. no
default choice: [1]:
2
Program exits now.
```

Now check the status of CW agent.

/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status

OR

systemctl status amazon-cloudwatch-agent

```
root@ip-172-31-52-40:/home/ubuntu# /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
{
  "status": "stopped",
  "starttime": "",
  "configstatus": "not configured",
  "cwoc_status": "stopped",
  "cwoc_starttime": "",
  "cwoc_configstatus": "not configured",
  "version": "1.247347.6b250880"
}
root@ip-172-31-52-40:/home/ubuntu#
```

Status is stopped now, so start it & check status after that.

/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json

```
root@ip-172-31-52-40:/home/ubuntu# /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
****** processing amazon-cloudwatch-agent ******
/opt/aws/amazon-cloudwatch-agent/bin/config-downloader --output-dir /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d --download-source file:/opt/aws/amazon-cloudwatch-agent/bin/config.json --mode ec2 --config /opt/aws/amazon-cloudwatch-agent/etc/common-config.toml --multi-config default
Successfully fetched the config and saved in /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_config.json.tmp
Start configuration validation...
/opt/aws/amazon-cloudwatch-agent/bin/config-translator --input /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json --input-dir /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d --output /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml --mode ec2 --config /opt/aws/amazon-cloudwatch-agent/etc/common-config.toml --multi-config default
2021/05/04 05:59:30 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_config.json.tmp ...
Valid Json input schema.
I! Detecting run_as_user...
No csm configuration found.
No log configuration found.
Configuration validation first phase succeeded
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -schematest -config /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml
Configuration validation second phase succeeded
Configuration validation succeeded
amazon-cloudwatch-agent has already been stopped
Created symlink /etc/systemd/system/multi-user.target.wants/amazon-cloudwatch-agent.service → /etc/systemd/system/amazon-cloudwatch-agent.service.
root@ip-172-31-52-40:/home/ubuntu# /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a status
{
  "status": "running",
  "starttime": "2021-05-04T05:59:31+00:00",
  "configstatus": "configured",
  "cwoc_status": "stopped",
  "cwoc_starttime": "",
  "cwoc_configstatus": "not configured",
  "version": "1.247347.6b250880"
}
root@ip-172-31-52-40:/home/ubuntu#
```

Troubleshooting:

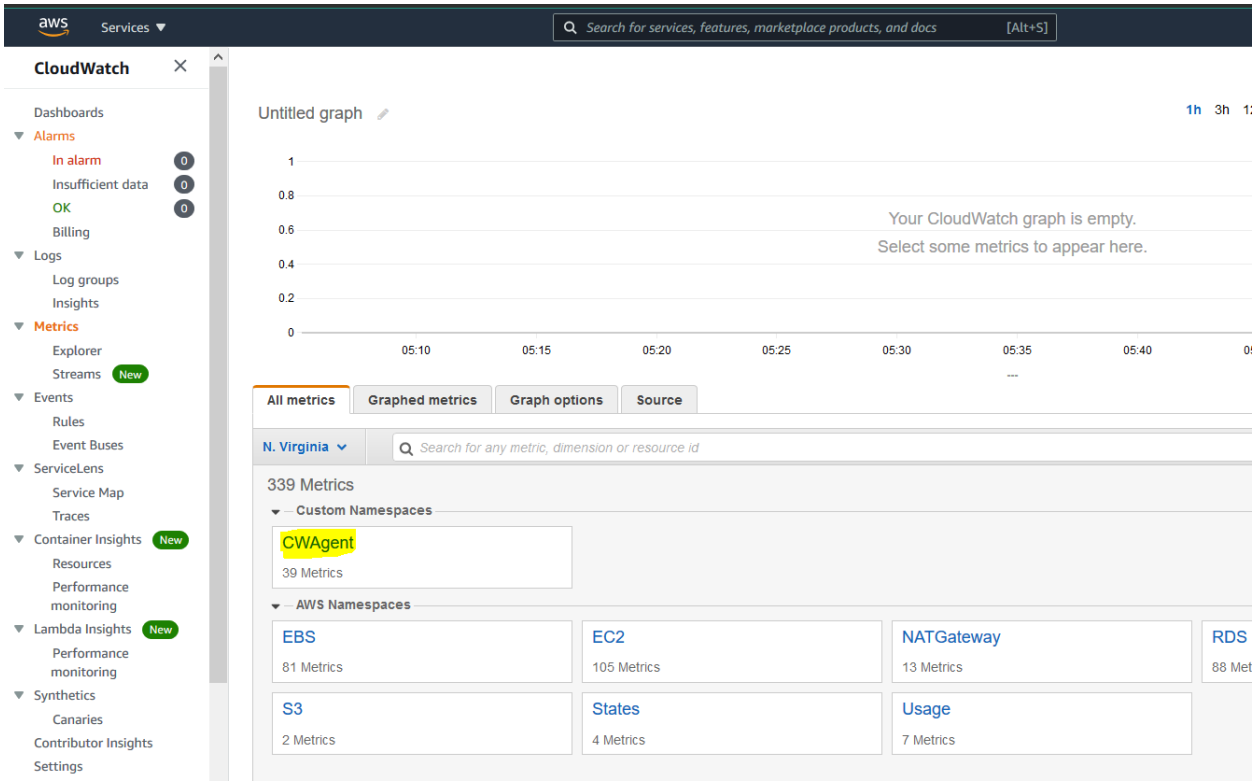
If config file for CW Agents emits error, execute below commands to setup collectD service for CW Agent.

sudo apt install collectd -y
mkdir -p /usr/share/collectd/

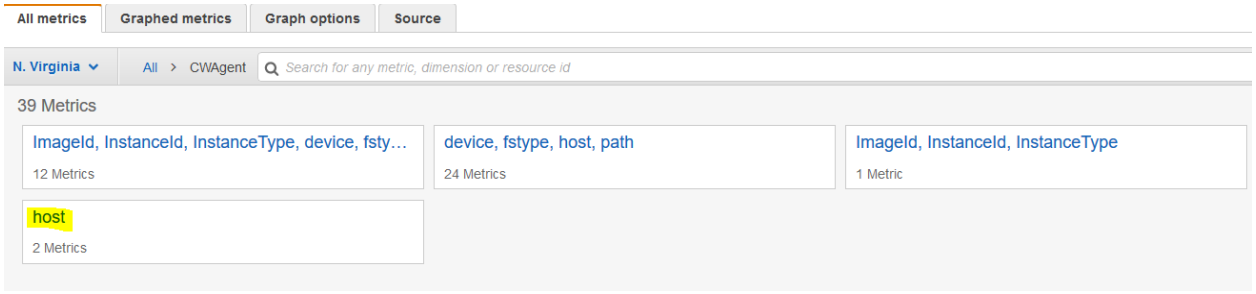
touch /usr/share/collectd/types.db

Step 5: Check Custom Metrics in AWS CloudWatch:

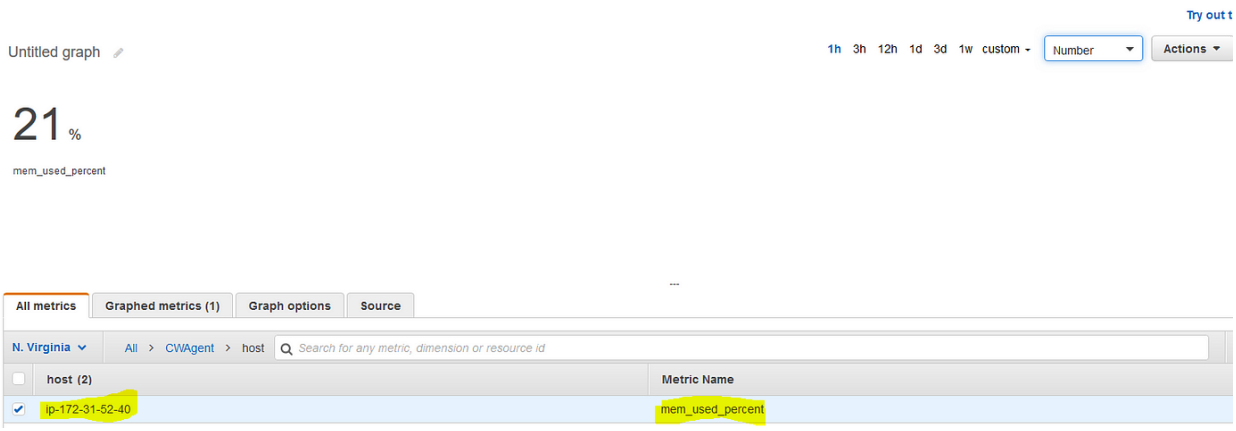
Custom Metrics:



Check inside a host for memory metrics:



Select the host according to the private IP of the instance & here you go with the memory metrics of the instance.



Step 6: Create a custom CW Dashboard and plot metrics on the created dashboard to monitor:

Now, since the metrics are visible on the CW Metrics, you can create a Custom Dashboard with Line graph, Bar graph, Pie chart visualization as per your monitoring requirements.

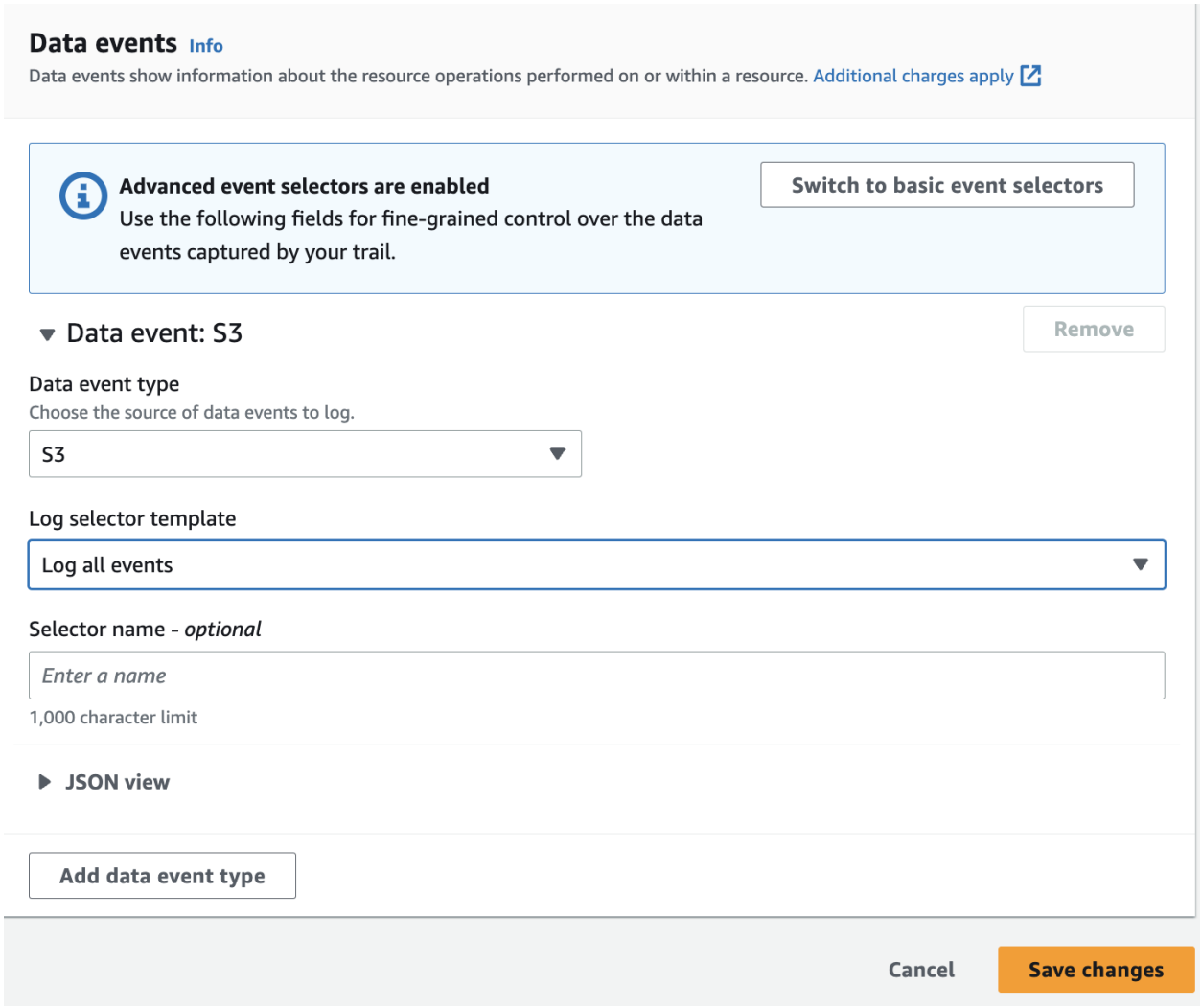
Select the metrics of your choice (which you want to monitor on the custom dashboard) under CWAgent Custom namespaces and click on Actions → Add to Dashboard → Select your custom dashboard where you want to capture and monitor these metrics.

6. Track user activities on your Cloud Environment using CloudTrail on any AWS Cloud Service.

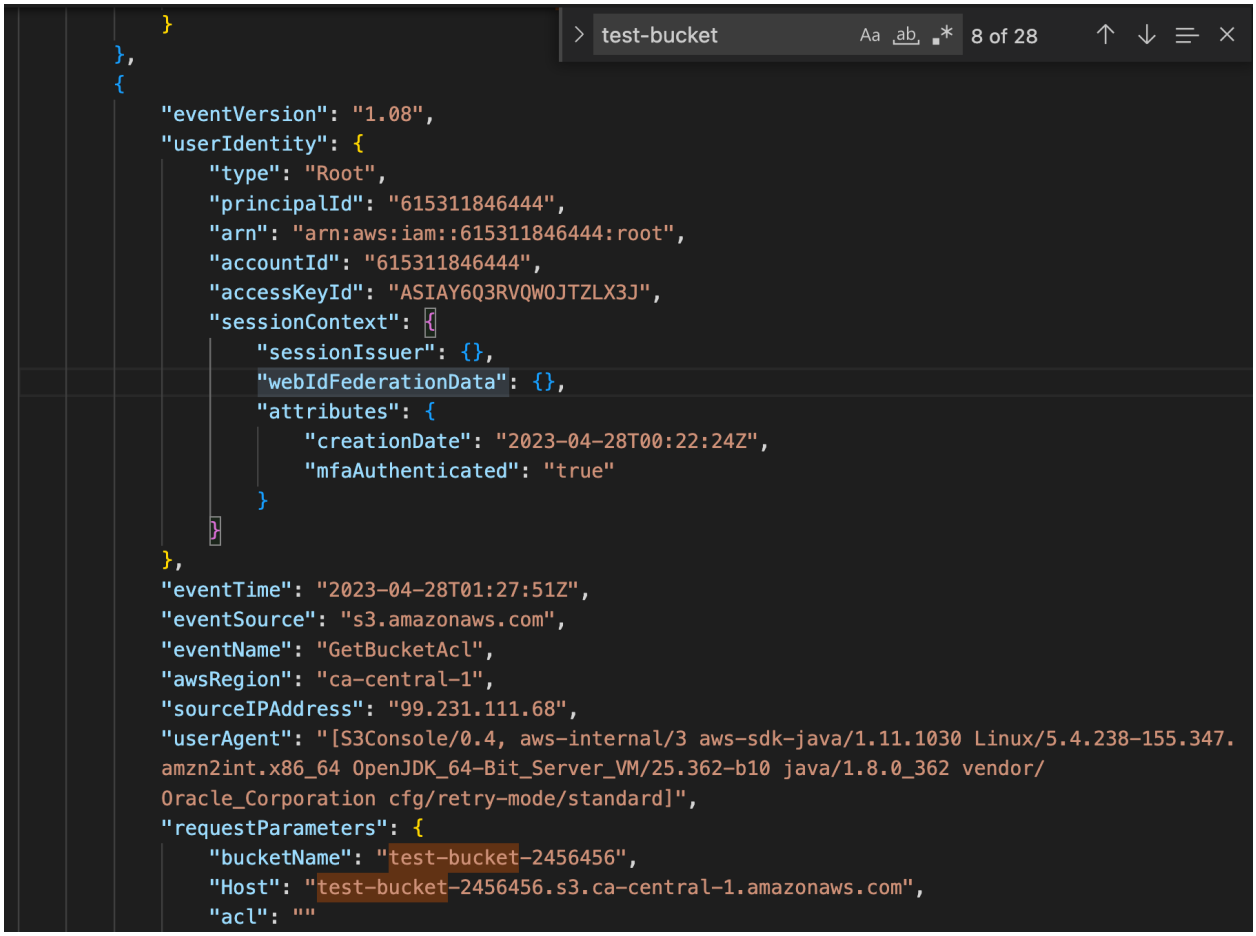
For example, let's assume a scenario: You are given the task to monitor all the user activities performed by the Cloud, DevOps, and Dev team on the AWS S3 bucket service. You can leverage CloudTrail to track API calls made via Console, CLI, SDK, or any other means.

So, all you need to do is just enter CloudTrail in the AWS service search box and go to the CloudTrail service. Click on **Create Trail** and then provide the basic details like Trail name, Storage bucket name, and folder, wether you want to log the user activities in CW Log Groups as well or not, and Tag values.

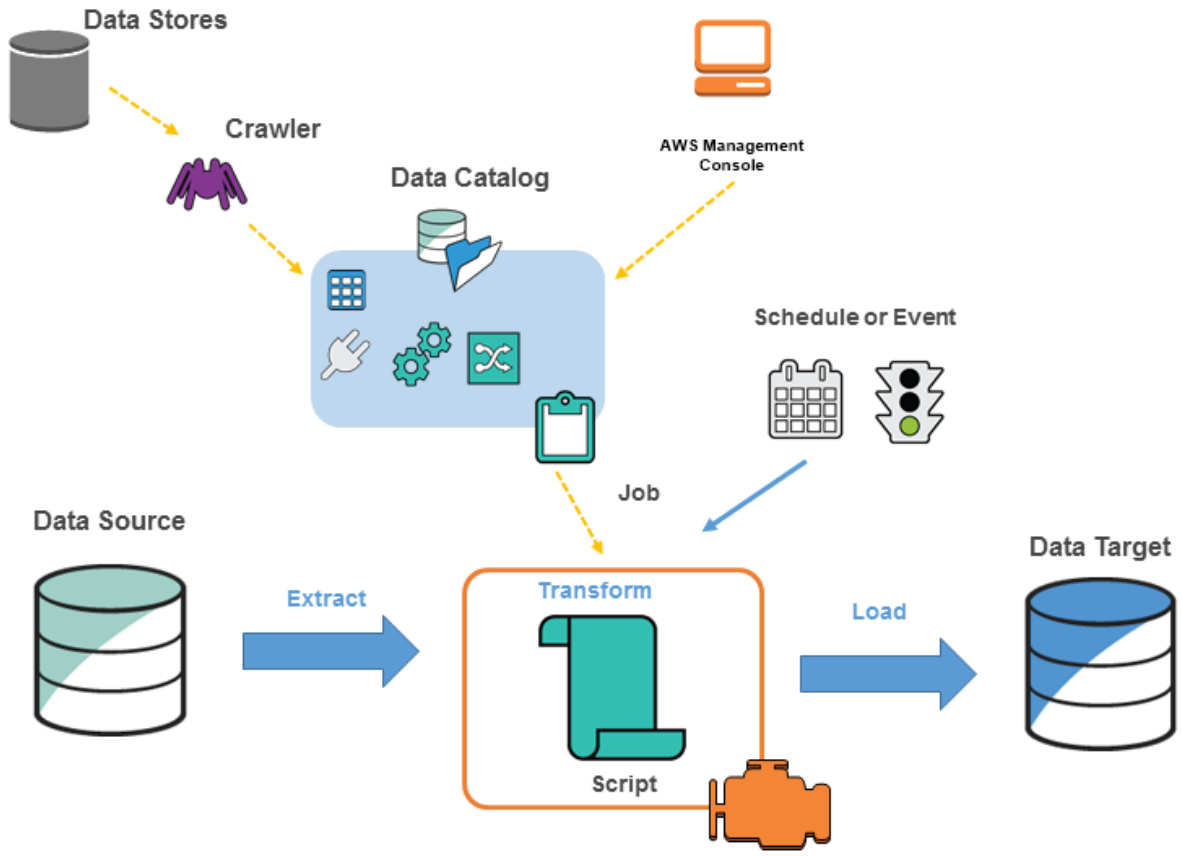
Once you create the Trail, go to the newly created Trail from the home page of CloudTrail and navigate to the **Data Events** section. Click on the Edit button and you can select as per the requirements such as Event type, Log Selector template (to select logging level), etc., and create a new Data event to capture in the Target Trail S3 bucket.



Finally, after applying these configurations, you can start receiving logs in the target S3 bucket or/and the AWS CW Log Group as per your selection within a few minutes as soon as the configured data event is triggered (In this case we selected S3 All events to capture) in a JSON format as below:



#13 Task: Walkthrough on an ETL (Extract, Load & Transform) Job using PySpark framework on AWS Glue with S3 Bucket, Glue Crawlers, and required IAM Roles. Also, Encrypt the data of the S3 bucket using a KMS CMK (Customer Managed Key) with a strong Cryptographic Algorithm.



1. Set S3 Buckets for source and destination data.

Glue can read data from a database or S3 bucket. For example, I have created an S3 bucket called glue-bucket-cloudspikes. Create two folders from the S3 console and

name them to read and write. Now create a text file with the following data and upload it to the read folder of the S3 bucket.

```
rank,movie_title,year,rating
1,The Shawshank Redemption,1994,9.2
2,The Godfather,1972,9.2
3,The Godfather: Part II,1974,9.0
4,The Dark Knight,2008,9.0
5,12 Angry Men,1957,8.9
6,Schindler's List,1993,8.9
7,The Lord of the Rings: The Return of the King,2003,8.9
8,Pulp Fiction,1994,8.9
9,The Lord of the Rings: The Fellowship of the Ring,2001,8.8
10,Fight Club,1999,8.8
```

2. Crawl the data source to the data catalog.

In this step, we will create a crawler. The crawler will catalog all files in the specified S3 bucket and prefix. All the files should have the same schema. In Glue crawler terminology the file format is known as a classifier. The crawler identifies the most common classifiers automatically including CSV, JSON, and parquet. Our sample file is in CSV format and will be recognized automatically.

- In the left panel of the Glue management console click Crawlers.
- Click the blue Add crawler button.
- Give the crawler a name such as glue-demo-cloudspikes-crawler.
- In Add a data store menu choose S3 and select the bucket you created. Drill down to select the read folder.
- In Choose an IAM role create a new one. Name the role for example glue-demo-cloudspikes-iam-role.
- In Configure the crawler's output add a database called glue-demo-cloudspikes-db.
- When you are back in the list of all crawlers, tick the crawler that you created. Click **Run** Crawler.

3. Query the captured data from S3 Bucket on Athena.

Go to the Database created and click on Table data and hit proceed to navigate to the Athena Query editor tool.

First, go to the Settings tab and set the Query result location as the destination S3 bucket with write specific folder. Then go back to the Editor tab in Athena and hit on the Run button.

This will execute the Athena query which looks like a SQL query to fetch the data from the source location which is Glue Database Table. This will also generate the data in the Query result location S3 bucket that you selected before as per the format you chose (JSON, CSV, etc).

4. Setup a Glue Job to perform ETL Operations.

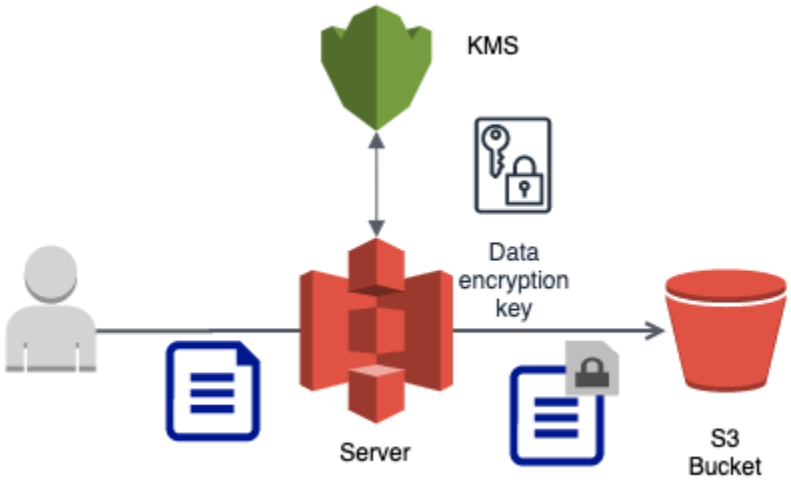
Go back to the Glue console and hit on Create button under the ETL Jobs → Visual ETL section. Name your job and configure the Data source and Data target S3 buckets as the same bucket but at different locations by clicking on the Visual canvas. For Data source make sure you select the read folder and for Data target select the write folder under glue-bucket-cloudspikes S3 Bucket.

Now, go to the Job details tab and select IAM Role as an existing role or create a new role that has full access to all the services used in this lab work. Go down and open the **Advanced Properties** section and name the Script filename as per your choice.

Once everything is done, you can hit on Run button by selecting the specific newly created Job. On the left-hand tab, you can navigate to the Job run monitoring tab and have a look at the Job status along with the CloudWatch Log Group screen that has Glue Job execution logs.

If the Glue job is successfully executed, you can see the generated data on the target S3 bucket under the write folder as per the configurations we set.

1. Encrypt data on S3 Bucket via Permission Policy.



In order to enforce object encryption, create an S3 bucket policy that denies any S3 Put request that does not include the x-amz-server-side-encryption header. There are two possible values for the x-amz-server-side-encryption header: AES256, which tells S3 to use S3-managed keys, and aws:kms, which tells S3 to use AWS KMS–managed keys. Bucket Policy will look like this

```
{
    "Version": "2012-10-17",
    "Id": "PutObjPolicy",
    "Statement": [
        {
            "Sid": "DenyIncorrectEncryptionHeader",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<bucket_name>/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "AES256"
                }
            }
        },
        {
```

```
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<bucket_name>/*",
            "Condition": {
                "Null": {
                    "s3:x-amz-server-side-encryption": true
                }
            }
        }
    ]
}
```

Now if we try to upload the object to S3 bucket without encryption it should fail:

$ aws s3 cp testingbucketencryption s3://mytestbuclet-198232055
upload failed: ./testingbucketencryption to s3://mytestbuclet-198232055/testingbucketencryption An error occurred (AccessDenied) when calling the PutObject operation: Access Denied

Let try it with encryption enabled:

$ aws s3 cp testingbucketencryption s3://mytestbuclet-198232055 --sse AES256
upload: ./testingbucketencryption to s3://mytestbuclet-198232055/testingbucketencryption