❖ **Title:** Intelligent EBS Volume Optimization Using Lambda, CloudWatch, SNS, DynamoDB & Step Functions.

❖ **Objective:**

Build a serverless automation pipeline that intelligently monitors EBS volumes, identifies gp2 volumes, and converts them to gp3, with full logging, alerting, and audit trail. This project reinforces knowledge of event-driven architecture, monitoring, notifications, data logging, and state orchestration.

This ensures:

1. Cost Optimization
2. Performance Improvement
3. Operational Efficiency
4. Security & Compliance
5. Disaster Recovery (via snapshots/rollback)

❖ **Steps:**

We are creating an automated system that continuously monitors EBS volumes, detects gp2 volumes, and converts them to gp3 with built-in logging, alerts, and rollback for safe and efficient operations.

1. **Launch an EC2 Instance with gp2 Volume**

   Go to EC2 Console → Launch Instance

   - Login to the AWS Management Console. Here region is Asia Pacific (Osaka).
   - Navigate to **EC2 → Instances → Launch Instance**.
   - Configure the instance with the following details:
     - **Name** - EBS-Demo-Instance
     - **AMI** - Amazon Linux 2 (Free Tier Eligible)
     - **Instance Type** - t2.micro
     - **Key Pair** - Select existing or create a new one.
     - **Storage** - keep default root volume (usually gp3).
   - Launch EC2.

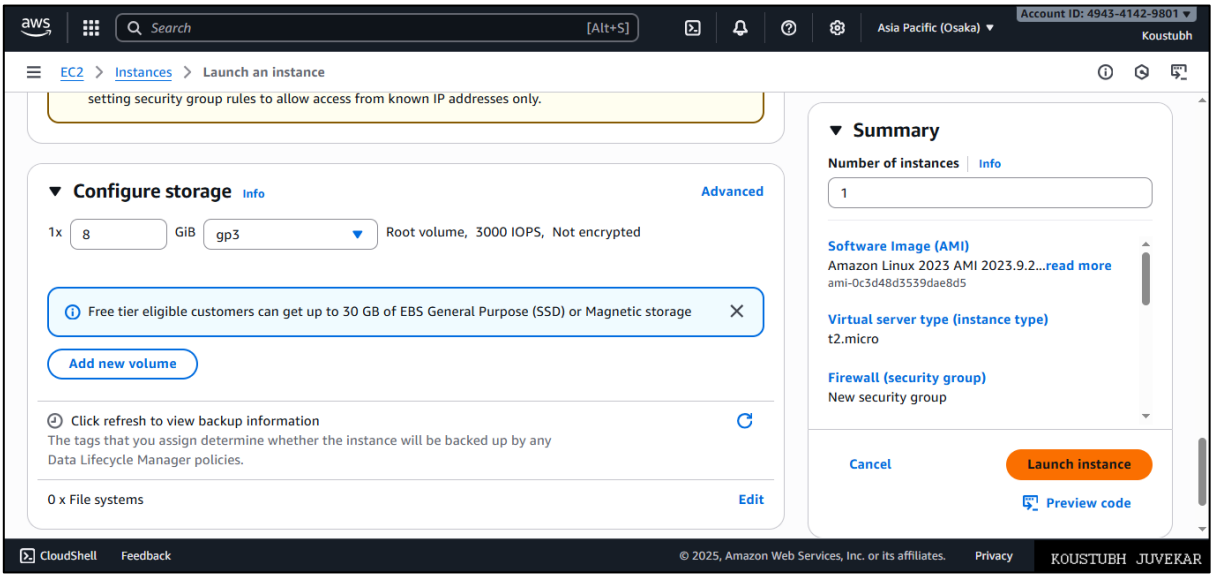

Image 1 : *EC2 launch with gp3 (Default)*

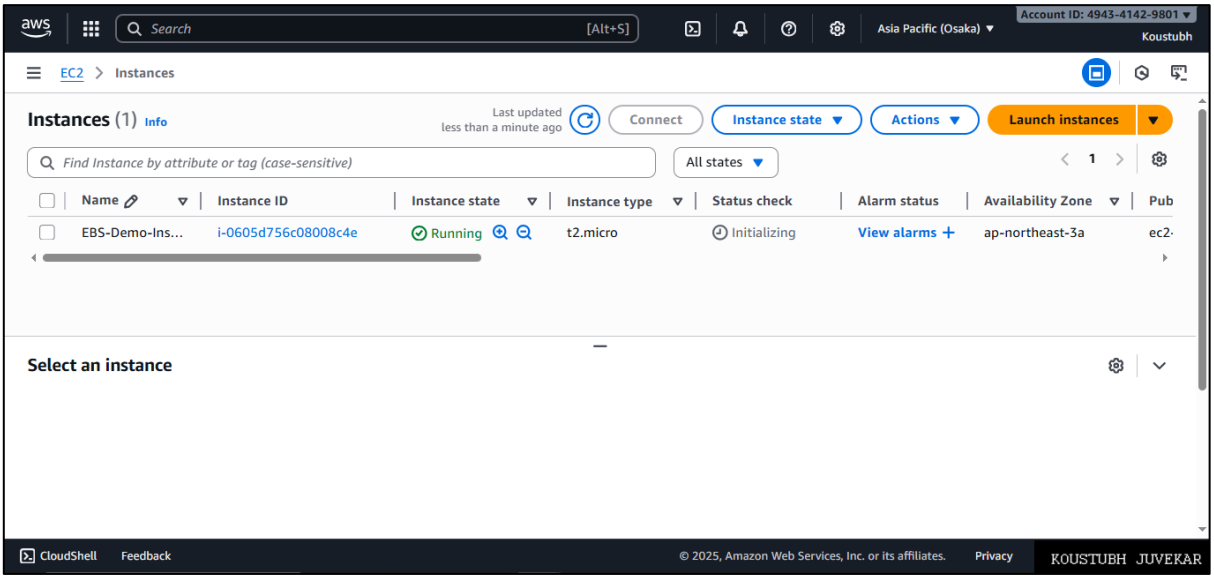

Image 1.1 : *EC2 launched with EBS* ***gp3***

After launch, create an extra volume:

- Availability Zone: same as your instance (important!).
- Go to Elastic Block **Store → Volumes → Create Volume.**
  - **Volume Type** - General Purpose SSD (gp2)
  - **Size** - 10 GiB
  - **Name** - EBS-Demo-Volume *(You can name volume from tag option.)*

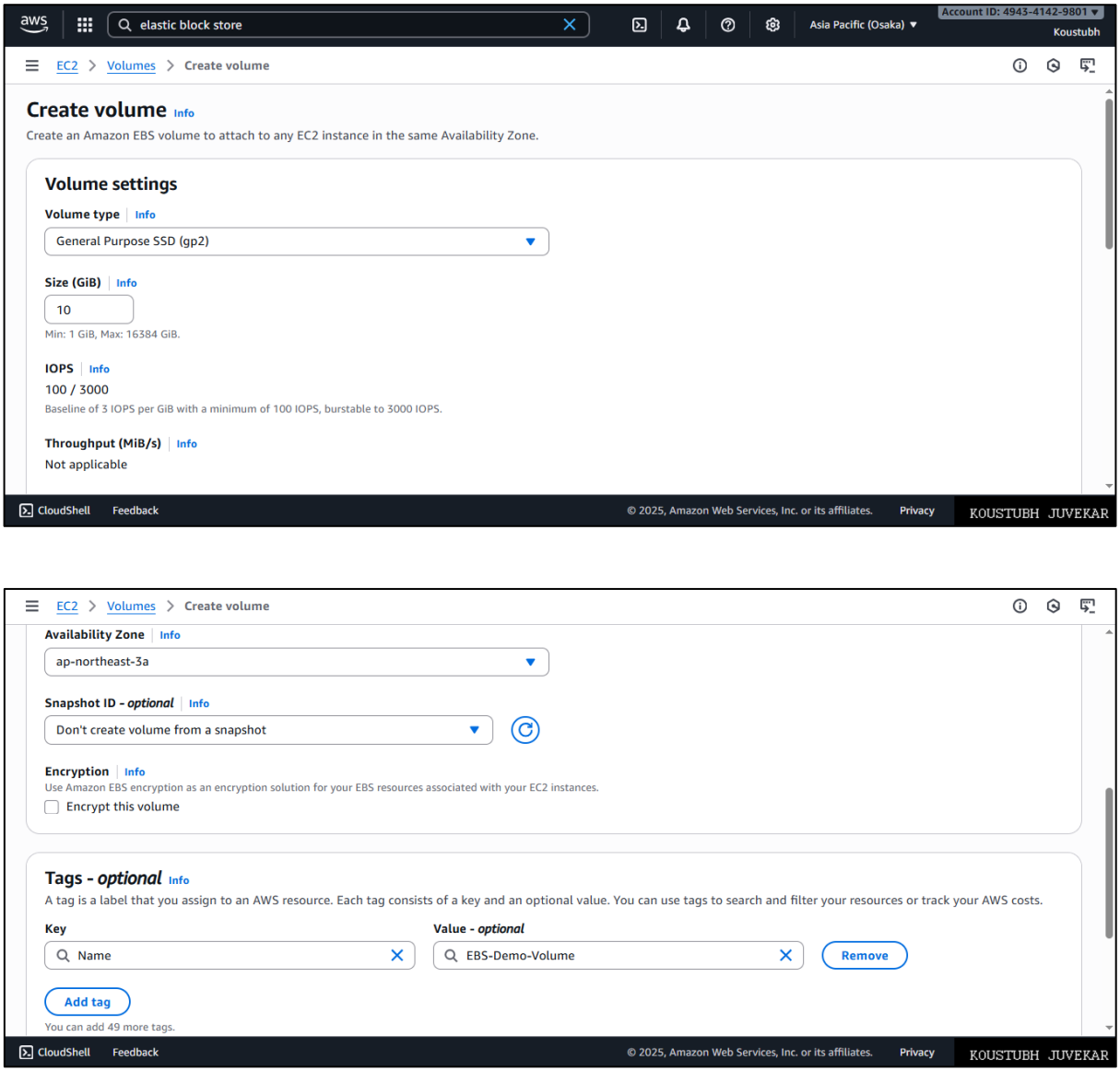

Image 1.2 : *Elastic block storage - Volumes - Create new volume.*

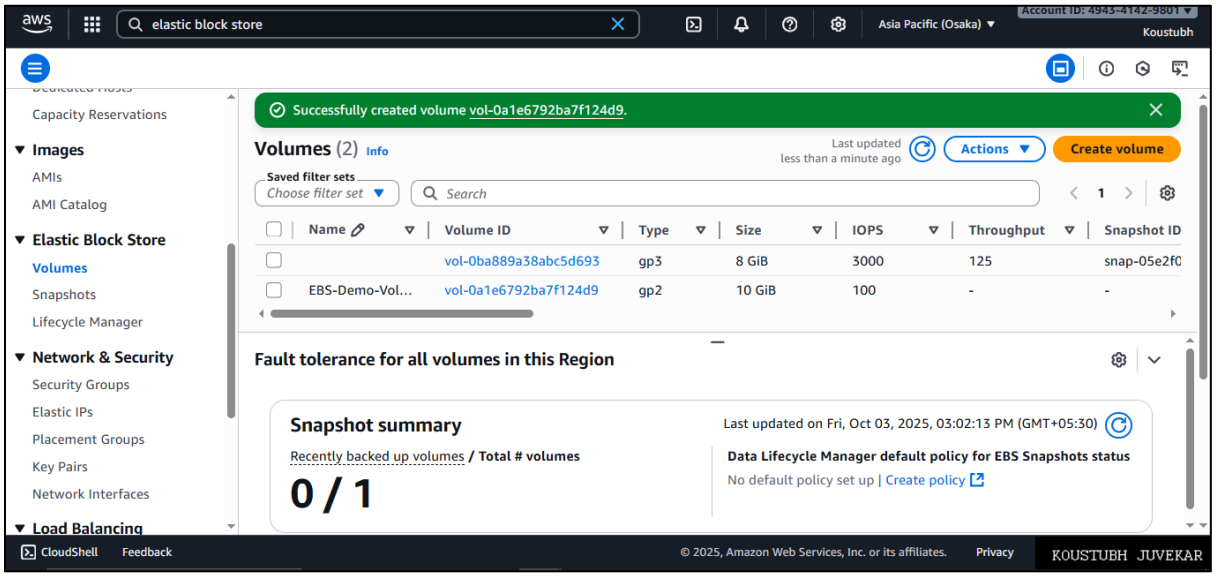

Image 1.3 : *Elastic block storage - Volumes - Create new volume*

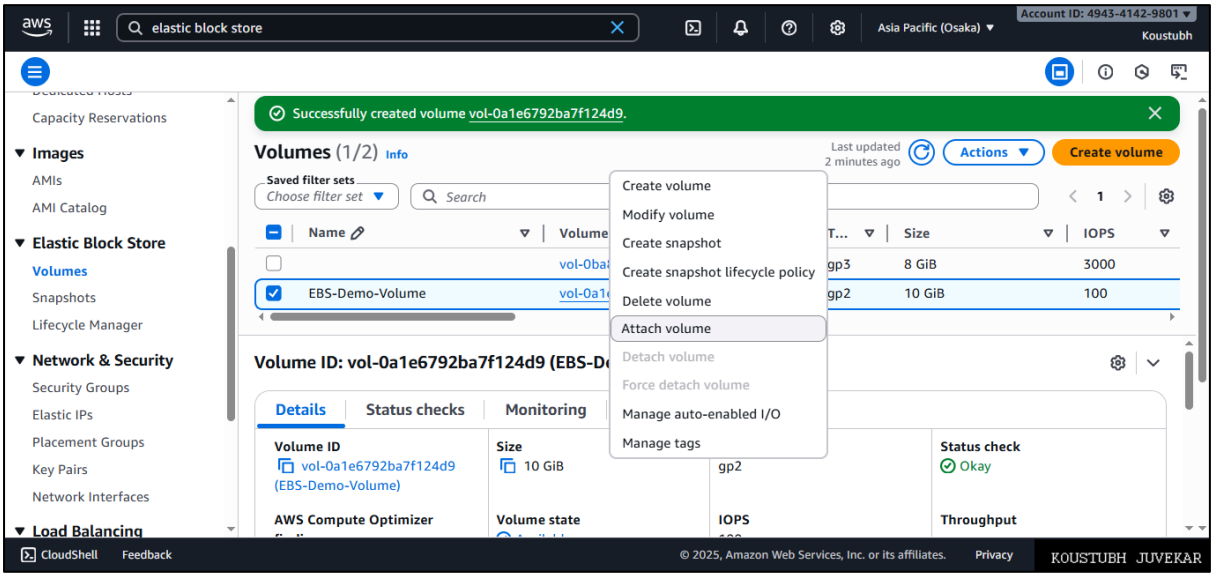

Image 1.4 : *Volumes created -- 1 default(gp3) - 1 created (gp2)*

Attach this new volume to your instance.

- Right-click → Attach Volume → select EBS-Demo-Instance.

Image 1.5 : *Attatch volume to the instance EBS-Demo-Instance*

- Select instance/EC2 EBS-Demo-Instance. from the list.
- Device name /dev/sdf
- Click on **Attach volume**

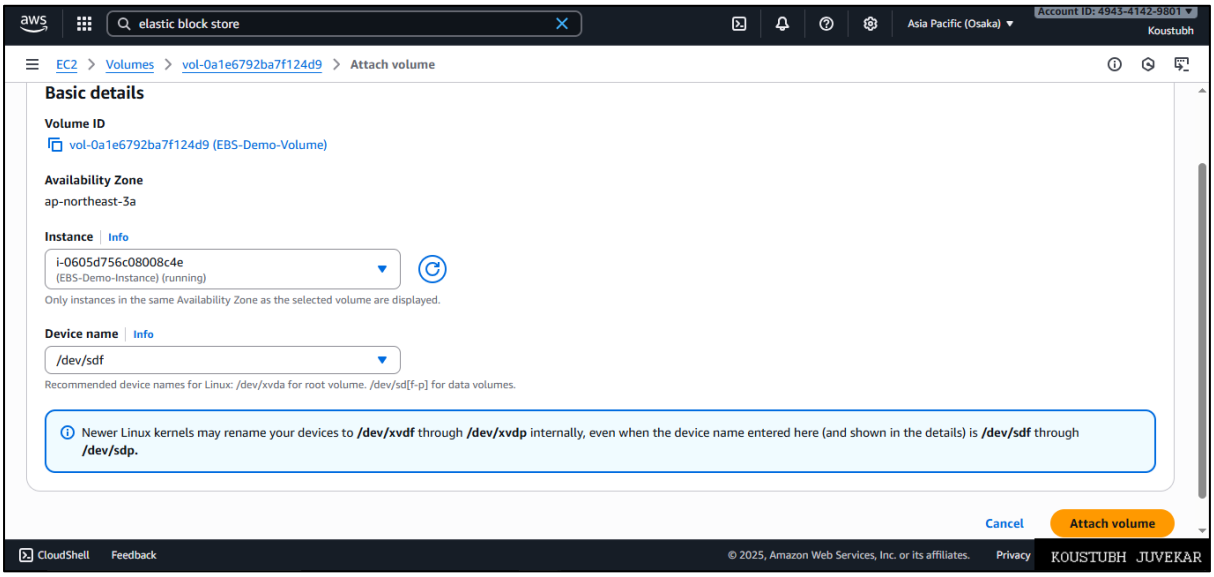

Image 1.6 : *Attach volume*

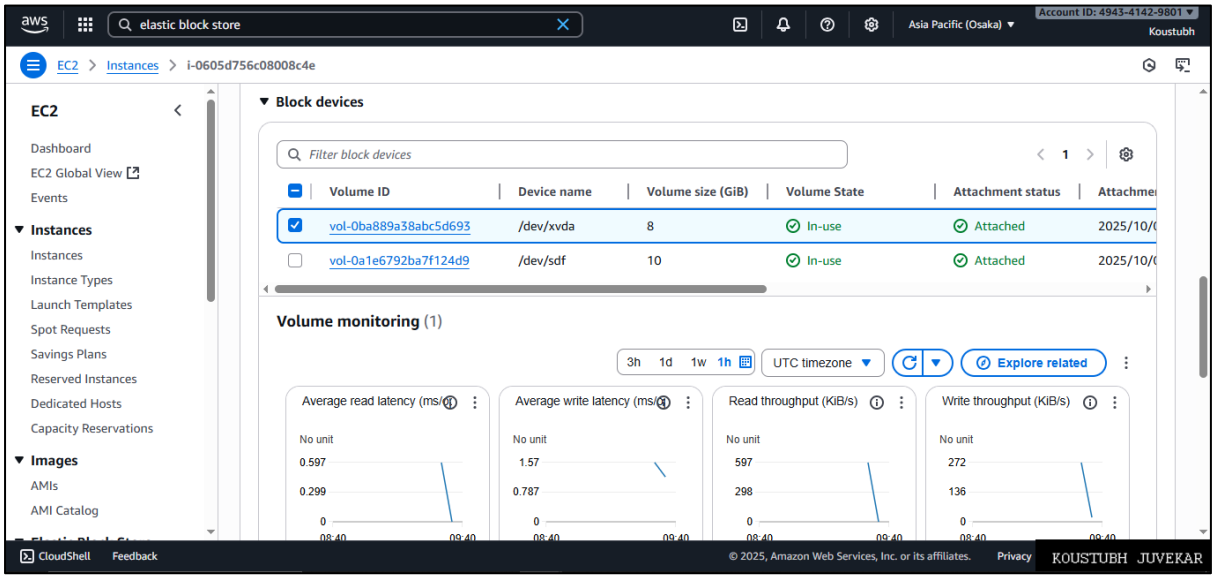

Image 1.6 : *Volumes attached to EC2*

2. Add a tag for auto-conversion

- Click on gp2 volume → Scroll down → Click on Tags → Click on Manage tags
- Tag the attached volume with:
  - **Key** - AutoConvert
  - **Value** - "true"

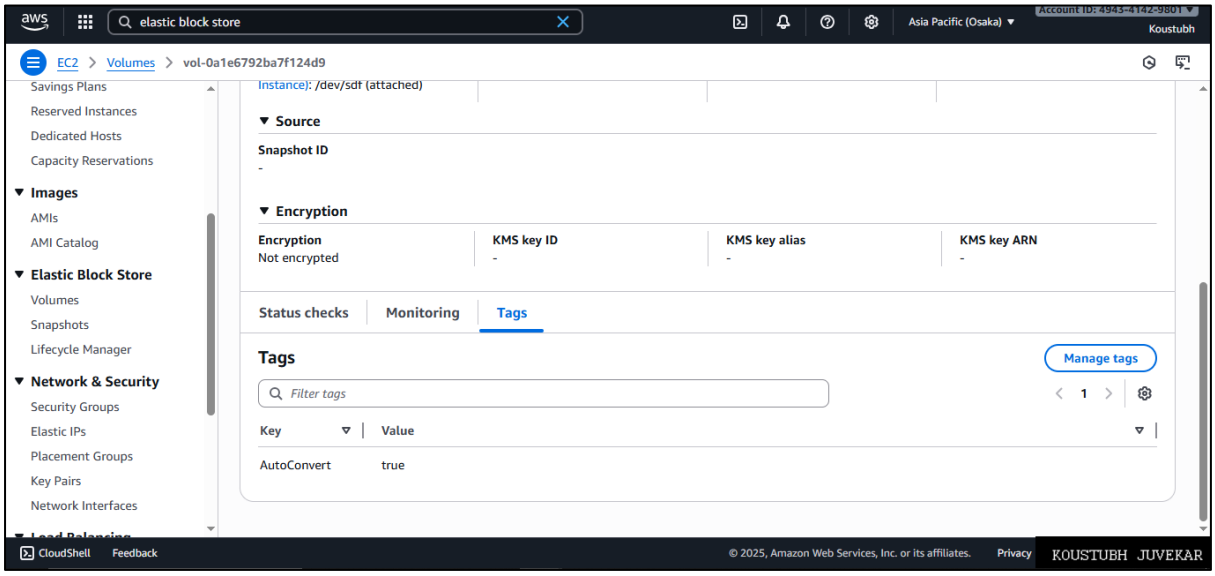

Image 2 : *Tagging Autoconvert= "true"*

3. Create DynamoDB table

- Go to **DynamoDB Console → Tables → Create Table**
  - **Table name** - EBSConversionLog
  - **Partition key** - VolumeId (String)
  - **Sort key** - Timestamp (String)
  - **Billing** - On-Demand
- Click on **Create table**

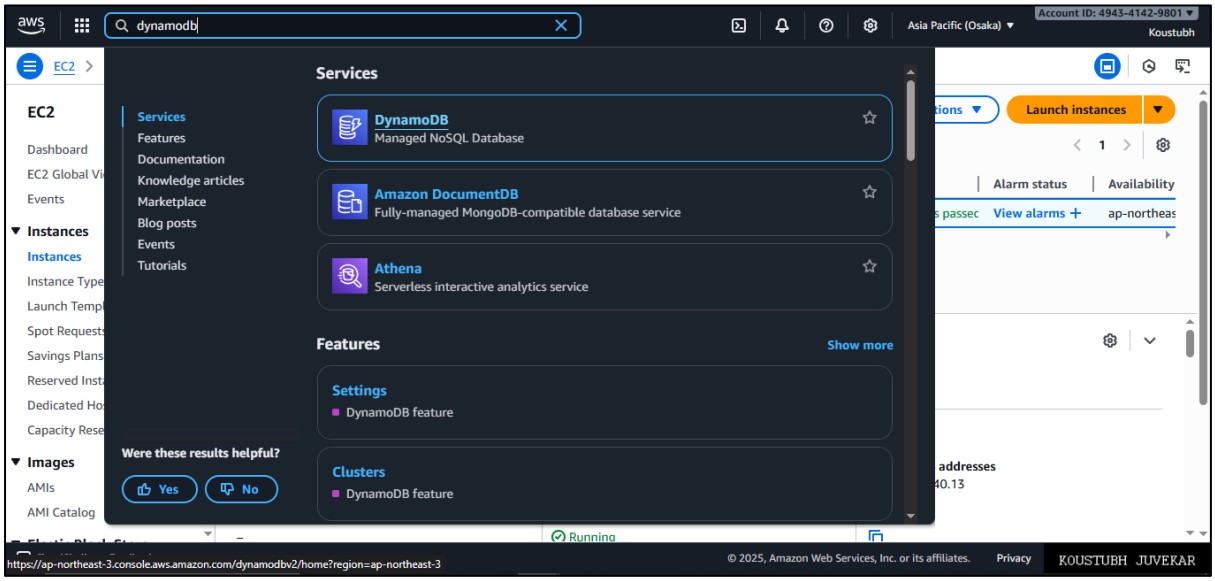

Image 3 : *Go to Dynamodb console*

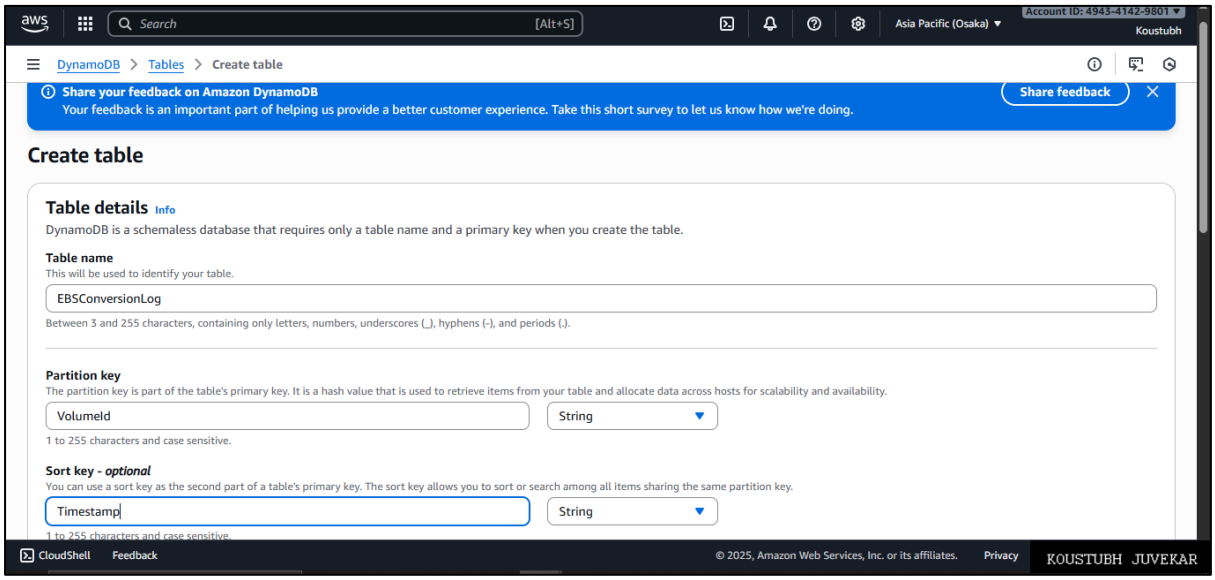

Image 3.1 : *Dynamodb - Create table*

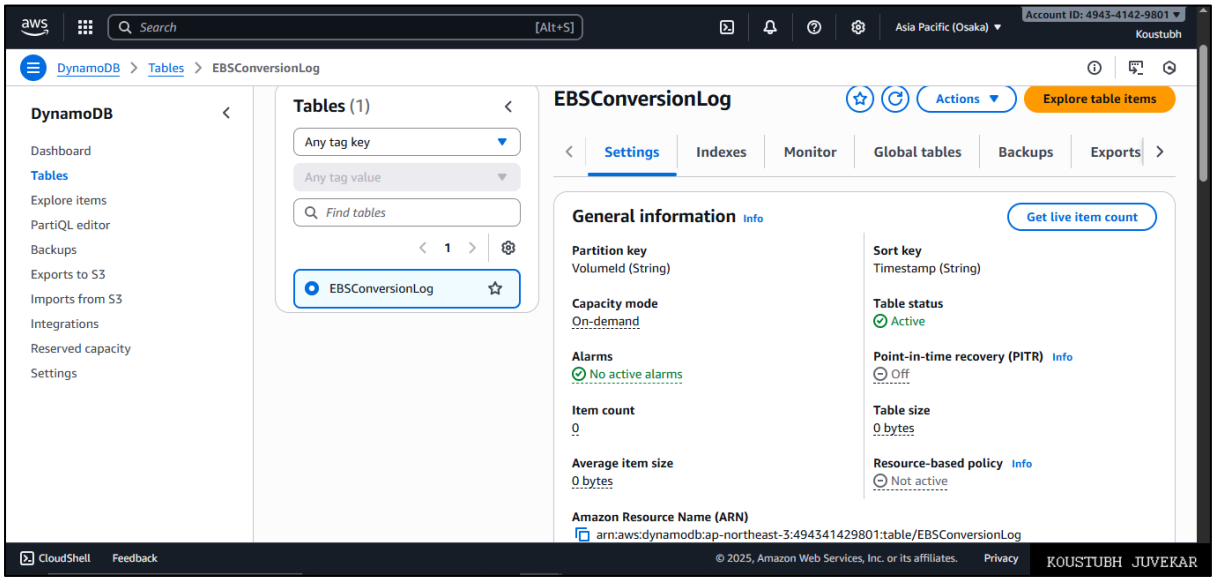

Image 3.2 : *Created table - Table details*

4. Create SNS topic

- Go to **SNS Console** → Topics → Create Topic

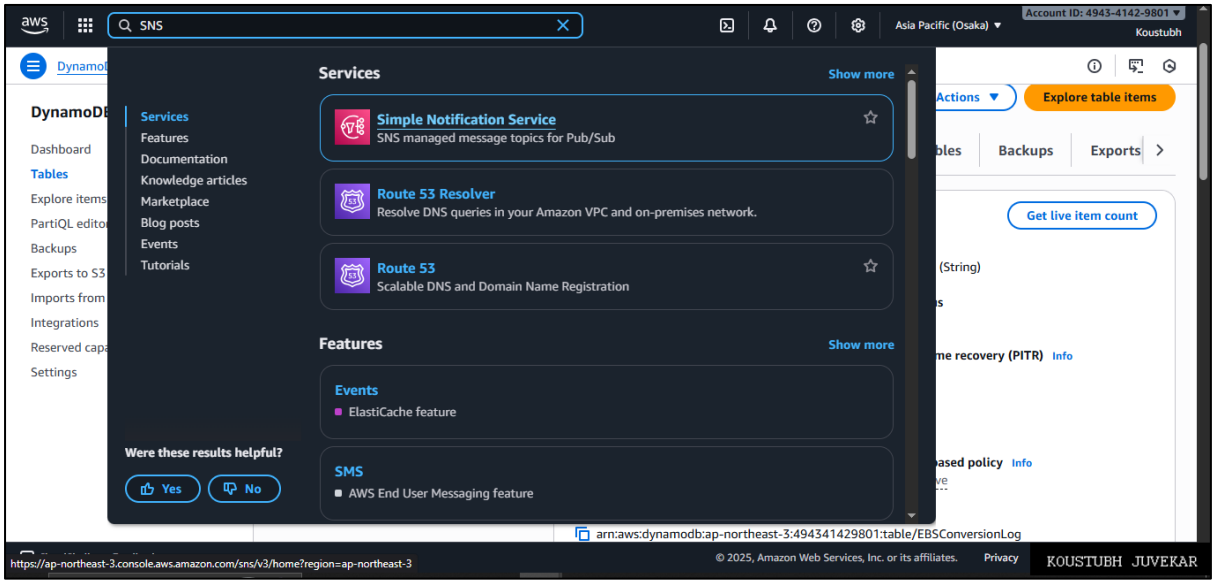

Image 4 : *SNS console*

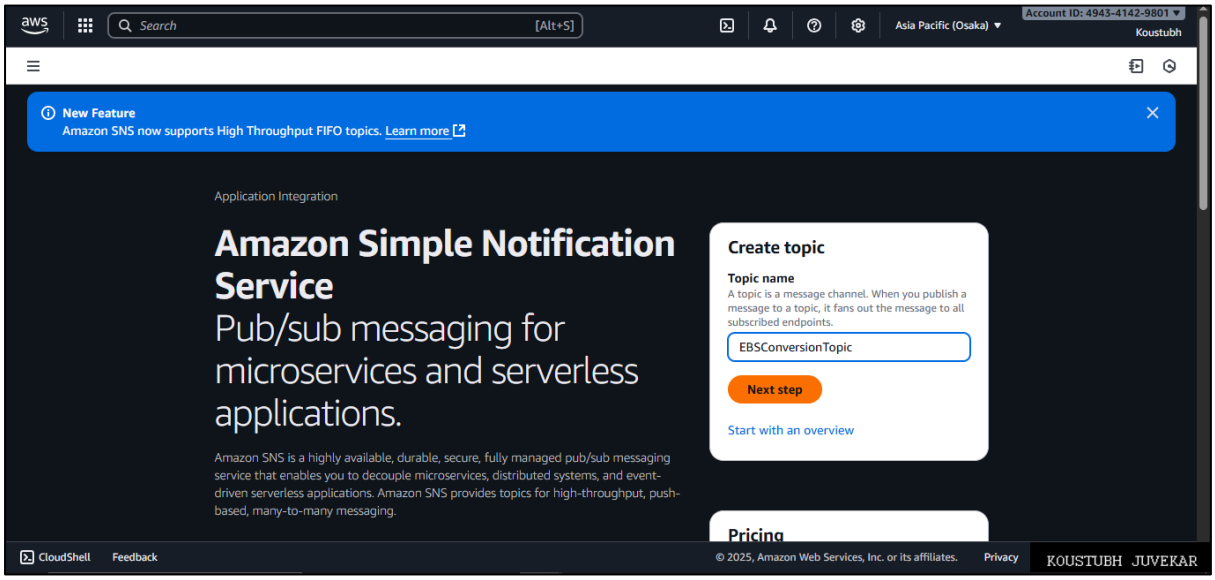

Image 4.1 : *SNS - Create Topic*

- Click on → **Create Topic**
  - **Type** - Standard
  - **Name** - EBSConversionTopic
  - **Display name - *optional*** - EBSConversionTopic
- Click on → **Create topic**

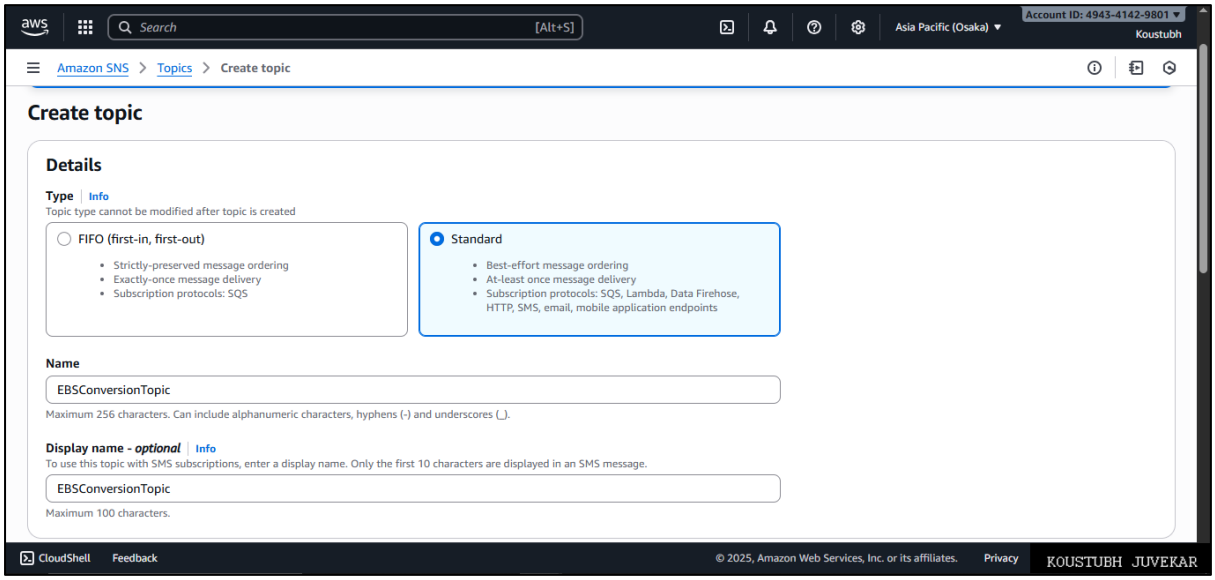

Image 4.2 : *SNS - Create Topic – options*

- Click on **Subscriptions** → click on **Create a subscription**
  - **Topic ARN** - `arn:aws:sns:ap-northeast-3:494341429801:EBSConversionTopic` *(It's the ARN of SNS topic.)*
  - **Protocol** - `Email`
  - **Endpoint** - `koustubhjuvekar07@gmail.com`
- Click on **Create subscription.**

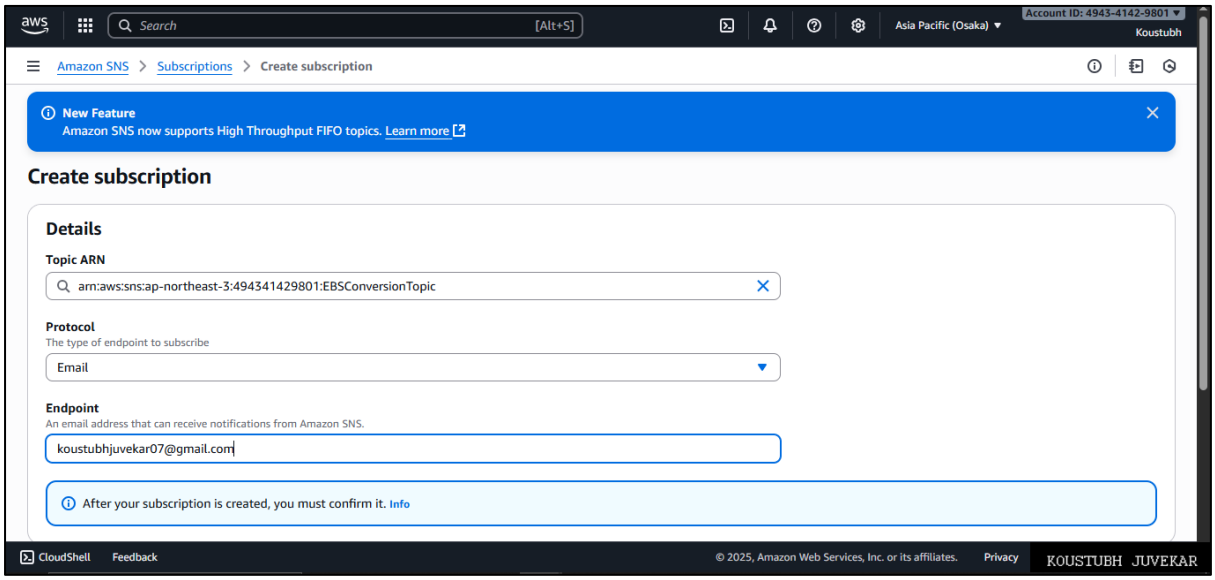

Image 4.3 : *SNS - Add Email*

- Confirm subscription in your mailbox.

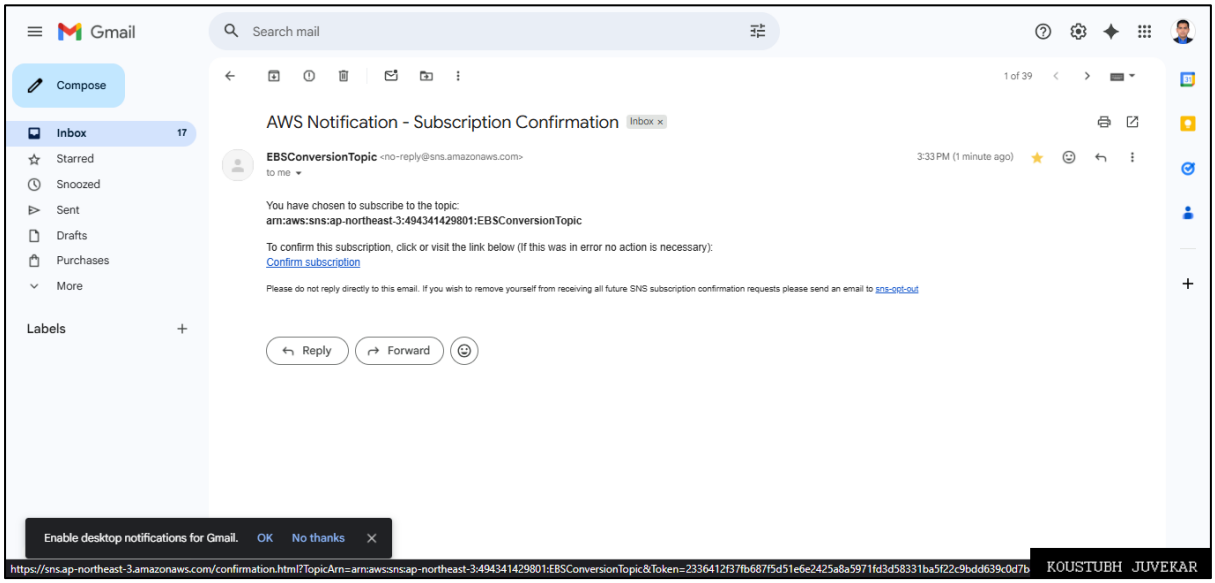

Image 4.4 : *SNS - Email confirm subscription*

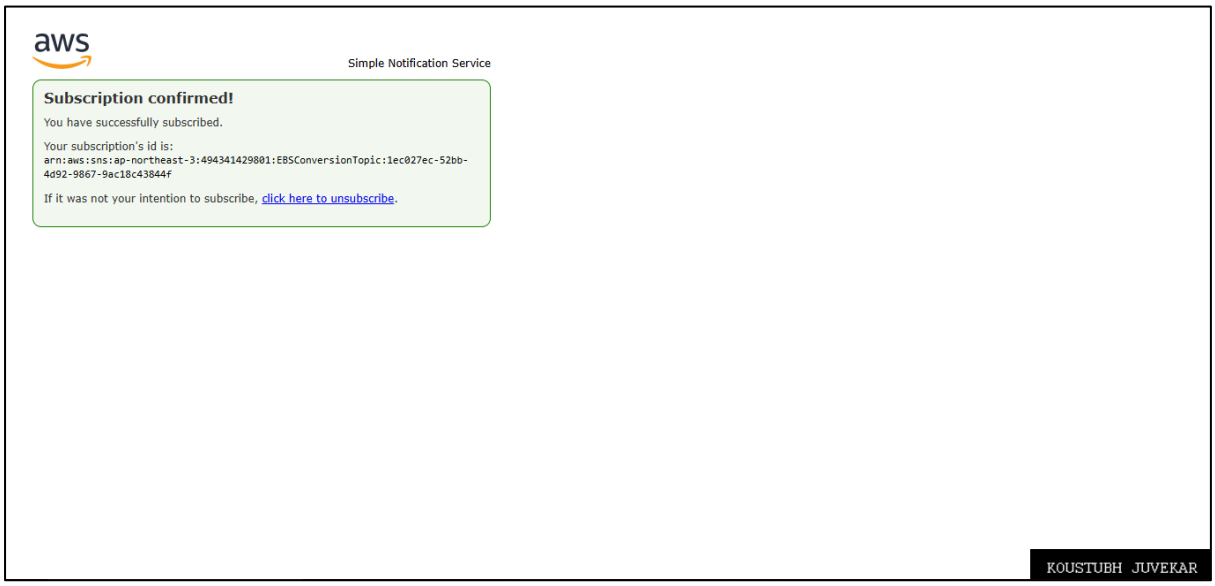

Image 4.5 : *SNS - Email confirmed*

5. **Create IAM Role for Lambda and Step Function**

   For Lambda (LambdaEBSRole)

   - Go to **IAM → Roles → Create Role**
     - **Trusted entity type** - AWS service
     - **Use case** → Service or use case - Lambda
     - **Click on** **Next**

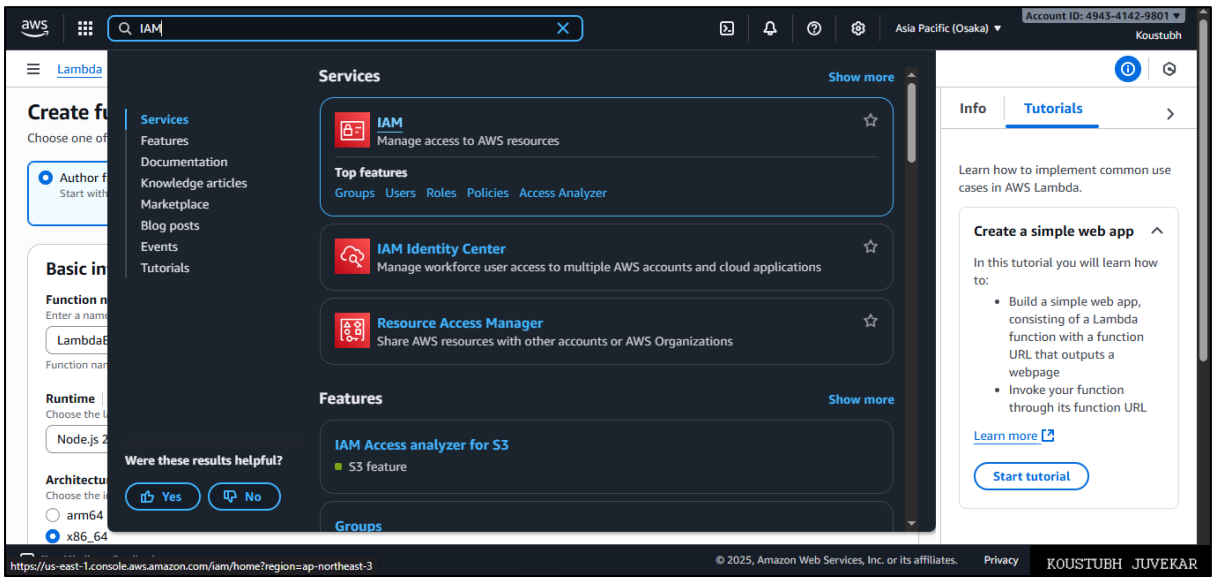

Image 5 : *Go to IAM console*

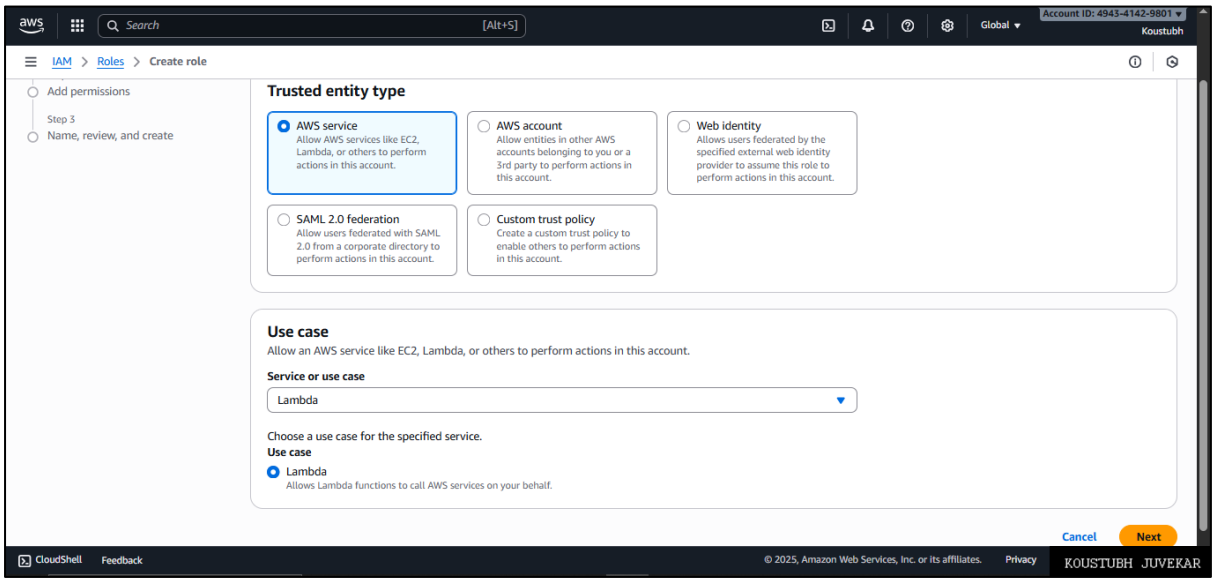

Image 5 : *IAM roles - create role page 1*

- Add Permissions → Permissions policies (1078) → Select following permissions(Search in box)
  - AmazonDynamoDBFullAccess
  - AmazonEC2FullAccess
  - AmazonSNSFullAccess
  - CloudWatchLogsFullAccess
- Click on Create role
- Name, review, and create
  - Role details →
    - **Role name** - LambdaEBSRole
    - **Description** - Allows Lambda function to call AWS service on your behalf.
- Click on **Create role.**

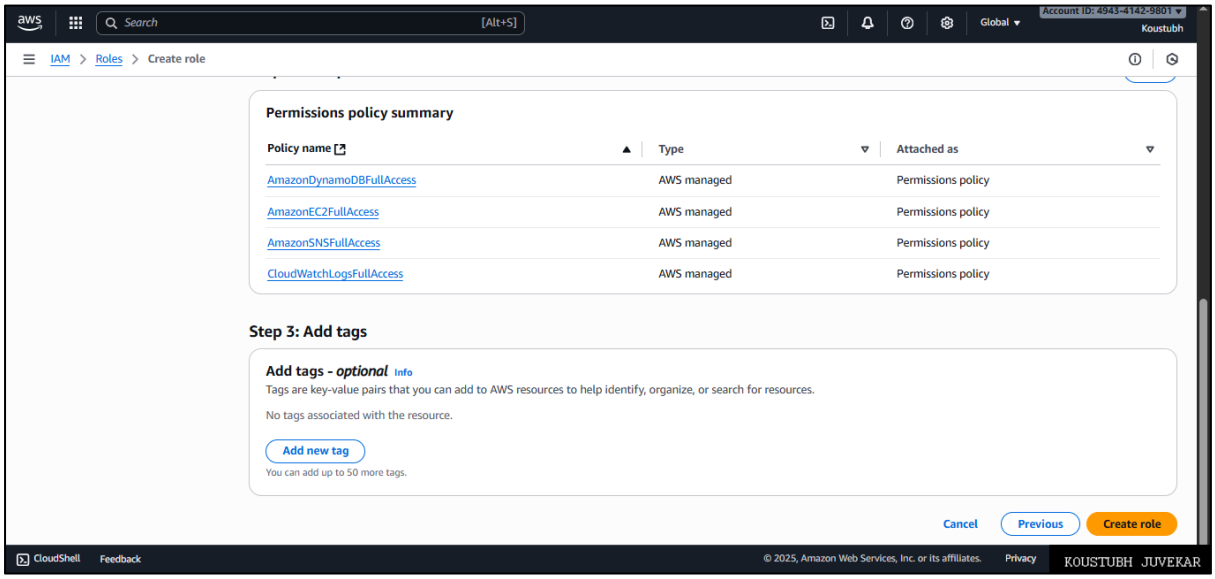

Image 5.1 : *IAM roles - create role page 3 - policies added previously*

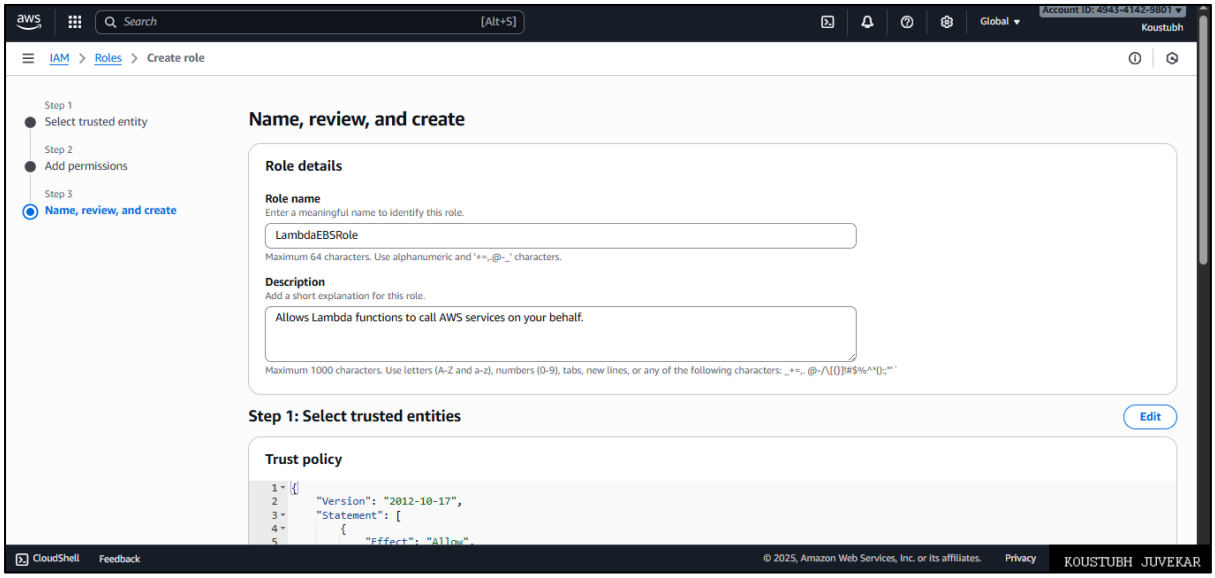

Image 5.2 : *IAM roles - IAM roles - create role page 3*

For Step Functions (StepFunctionsEBSRole)

- Go to **IAM → Roles → Create Role**
  - **Trusted entity type** - AWS service
  - **Use case** → Service or use case - Step Functions
  - **Click on** Next
- Add Permissions → Permissions policies (1078)
  - → Click on Set permissions boundary - optional
  - → Select Use a permissions boundary to control the maximum role permissions

- Search and select CloudWatchLogsFullAccess AWSLambdaRole policies.

- Name, review and create
  - Role details →
    - **Role name** - StepFunctionsEBSRole
    - **Description** - Allows Step Functions function to call AWS service on your behalf.
- Click on **Create role**

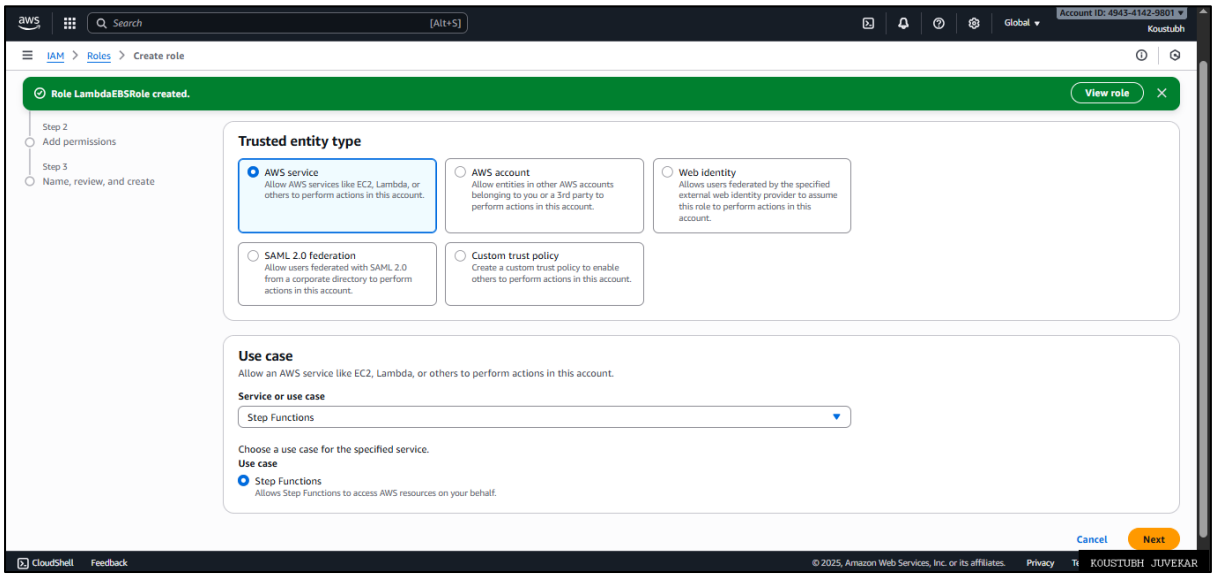

Image 5.3B : *IAM roles - create role for Stepfunction*

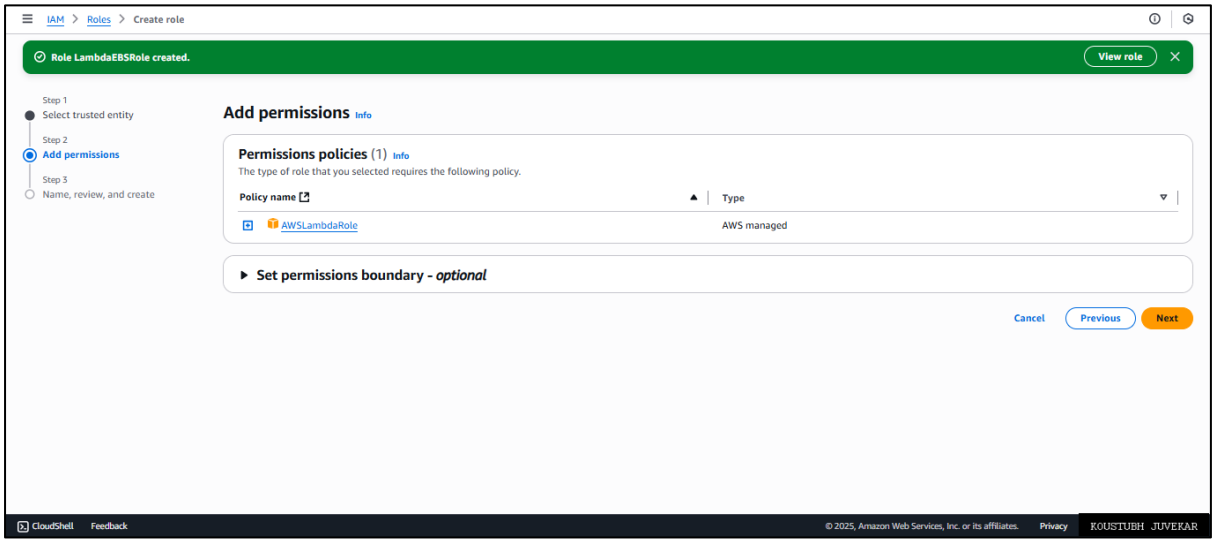

Image 5.4B : *IAM roles - Create role for Stepfunction - Attach AWSLambdaRole*

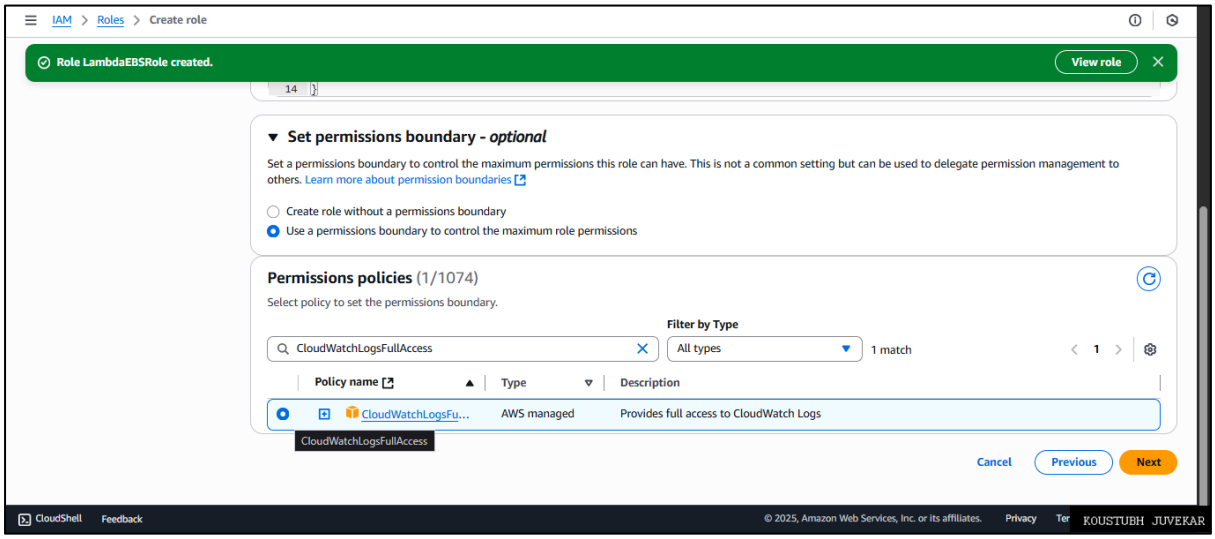

Image 5.5B : *IAM roles - create role for Stepfunction - Attach AWSLambdaRole - Use a permissions boundary to control the maximum role permissions*

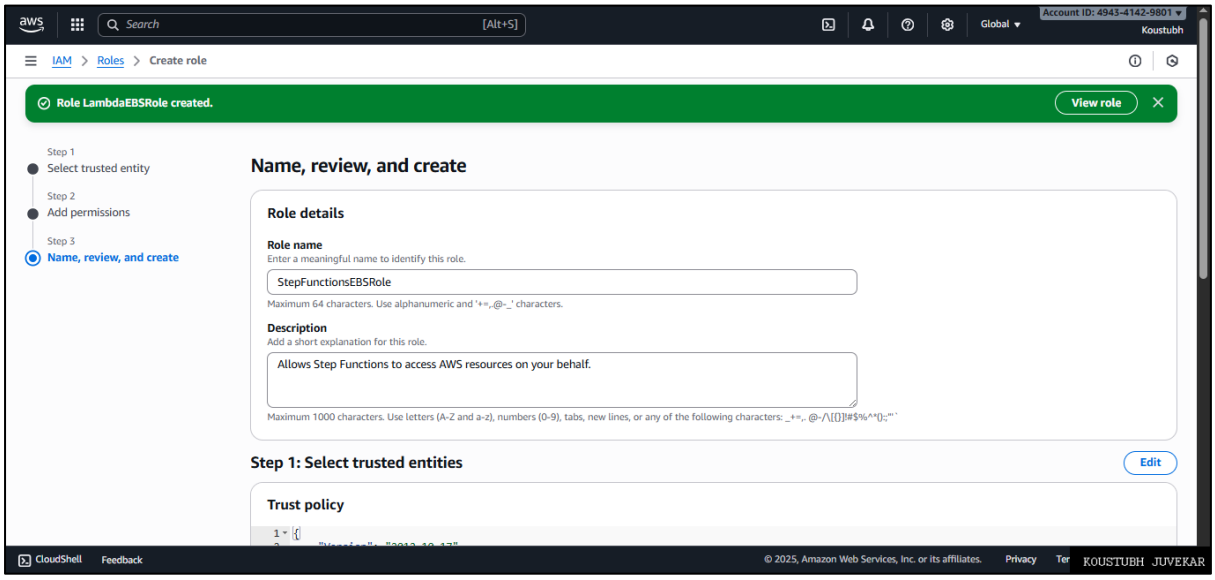

Image 5.6B : *IAM roles - create role for Stepfunction - Page 3*

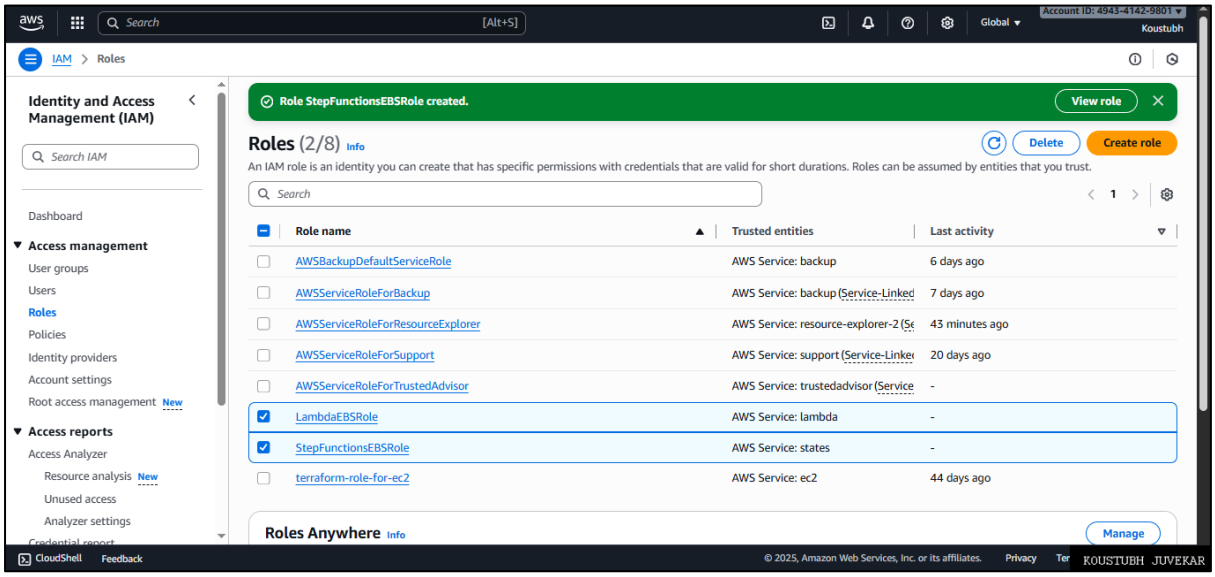

Image 5.7 : *IAM roles created.*

6. Create Lambda functions

EBSFilterLambda

- Go to **Lambda → Create function**

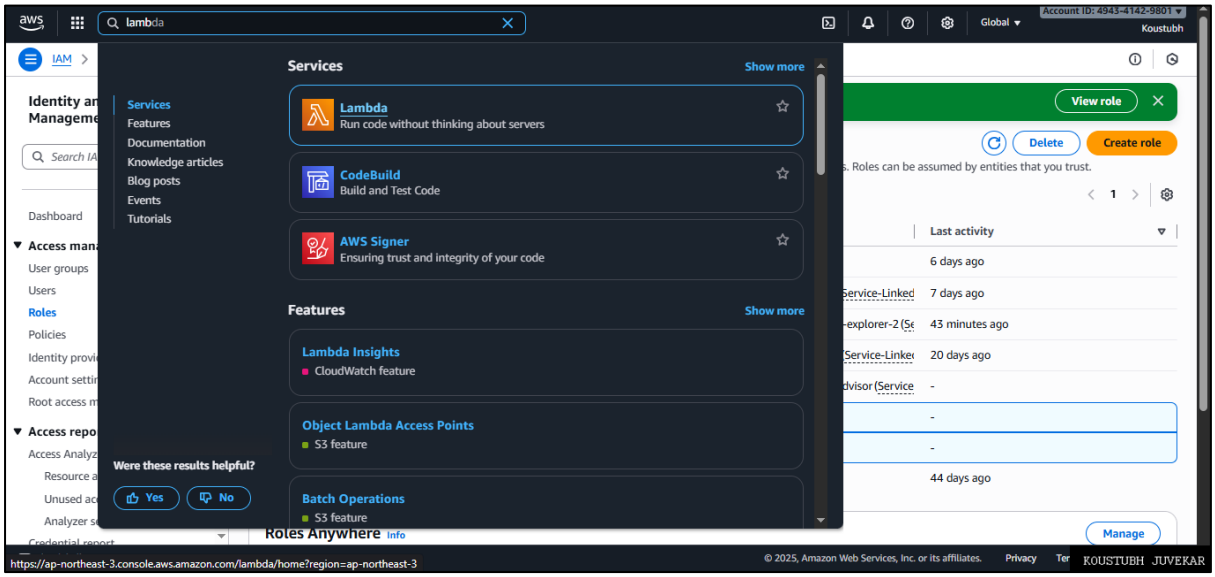

Image 6 : *Go to Lambda console.*

- Click on Create function → Author from scratch
- Basic information
  - Function name - `EBSFilterLambda`
  - Runtime - `Python 3.13`
- Permission → Change default execution role →
- Use an existing role →
  - Existing Role - `LambdaEBSRole`
- Click on **Create function**

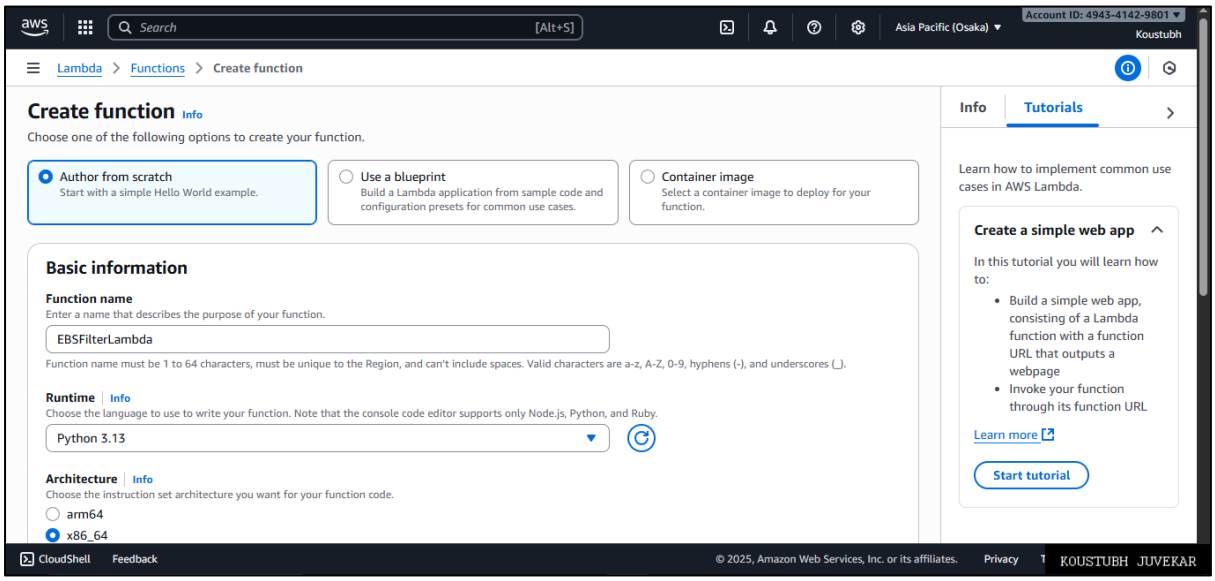

Image 6.1 : *Create Function - Options.*

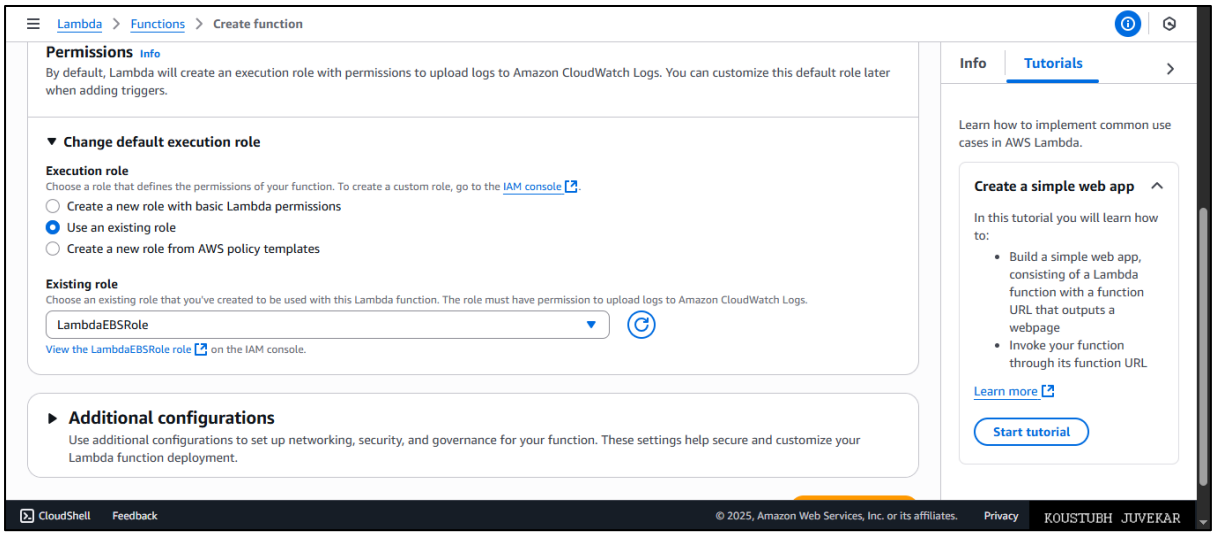

Image 6.2 : *Create Function - Existing role.*

- Add environment variable →
  - **Key** - DDB_TABLE
  - **Value** - EBSConversionLog
- Click on **Save**
- It will be as DDB_TABLE = EBSConversionLog

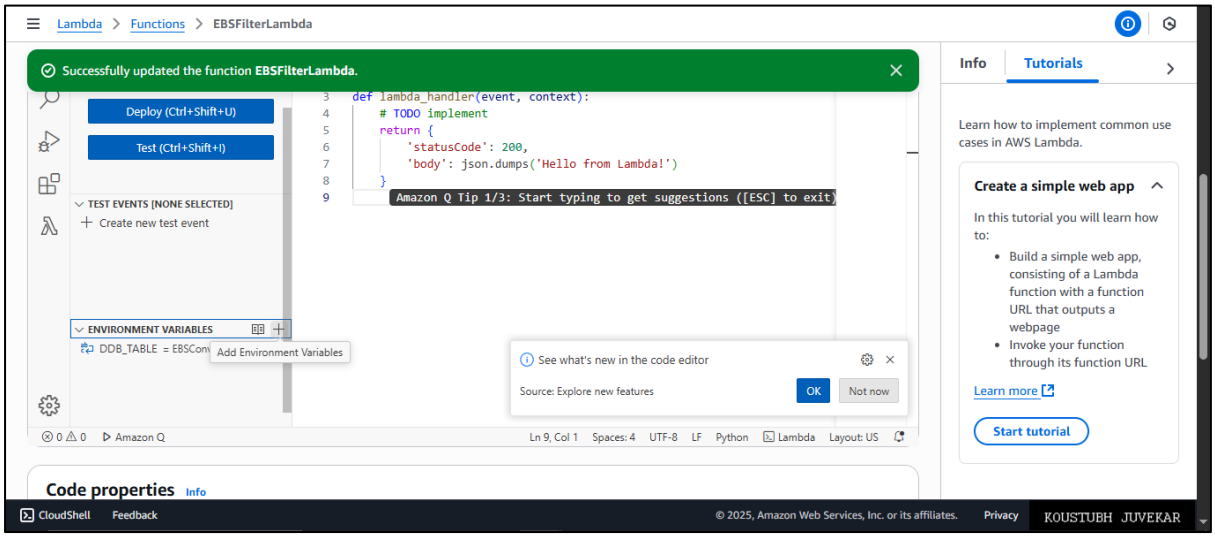

Image 6.3 : *Add environment variable.*

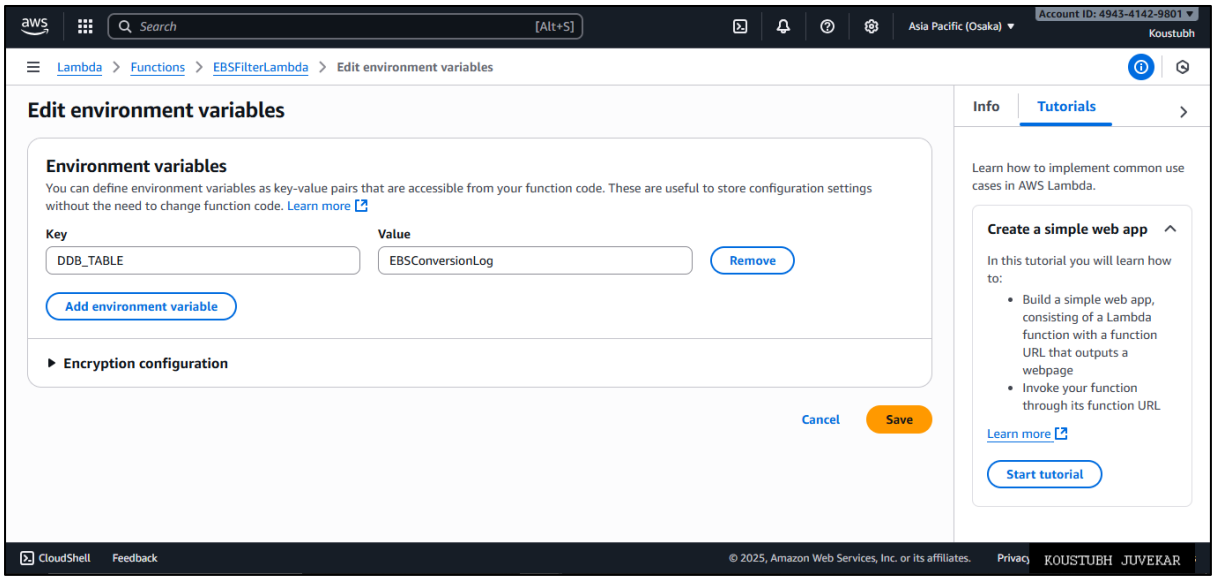

Image 6.4 : *Add environment variable.*

- Paste code for filter function *lambda_function.py*

```
import boto3
import os
from datetime import datetime

def lambda_handler(event, context):
    print("Starting EBS Filter Lambda")
    ec2 = boto3.client('ec2')
    dynamodb = boto3.client('dynamodb')

    try:
        # Get all gp2 volumes with AutoConvert=true tag
        response = ec2.describe_volumes(
            Filters=[
                {'Name': 'volume-type', 'Values': ['gp2']},
                {'Name': 'tag:AutoConvert', 'Values': ['"true"']}
            ]
        )

        volumes = response['Volumes']
        print(f"Found {len(volumes)} gp2 volumes with
AutoConvert=true")

        volume_ids = []

        # Process each volume
        for volume in volumes:
            volume_id = volume['VolumeId']
            print(f"Processing volume: {volume_id}")
            volume_ids.append(volume_id)

            # Record in DynamoDB
            dynamodb.put_item(
                TableName=os.environ['DDB_TABLE'],
                Item={
                    'VolumeId': {'S': volume_id},
                    'Timestamp': {'S':
datetime.utcnow().isoformat()},
                    'Status': {'S': 'PENDING'},
                    'Action': {'S': 'Identified'}
                }
            )
            print(f"Recorded {volume_id} in DynamoDB")

        # Return in simplified format for Step Functions
      return {
```

```
            'statusCode': 200,
            'Volumes': volume_ids
        }

    except Exception as e:
        print(f"Error: {str(e)}")
        return {
            'statusCode': 500,
            'Volumes': []
        }
```

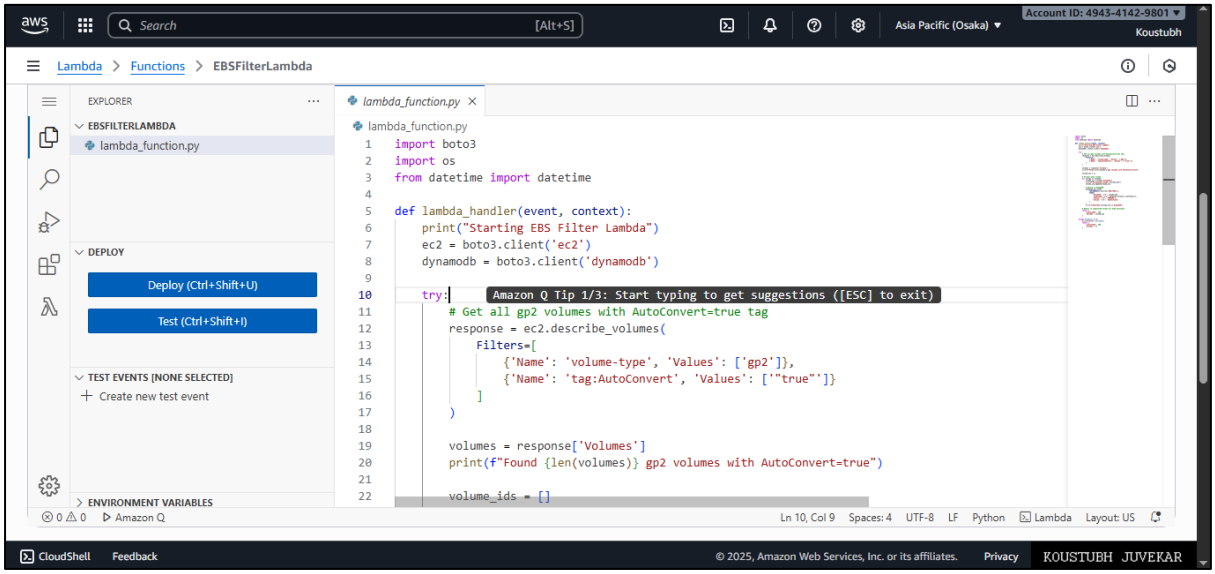

Image 6.5 : *Add code.*

EBSModifyLambda

- Go to **Lambda → Create function**
- Click on Create function → Author from scratch
- Basic information
  - **Function name** - EBSModifyLambda
  - **Runtime** - Python 3.13
- Permission → Change default execution role →
- Use an existing role →
  - **Existing Role** - LambdaEBSRole
- Click on **Create function**

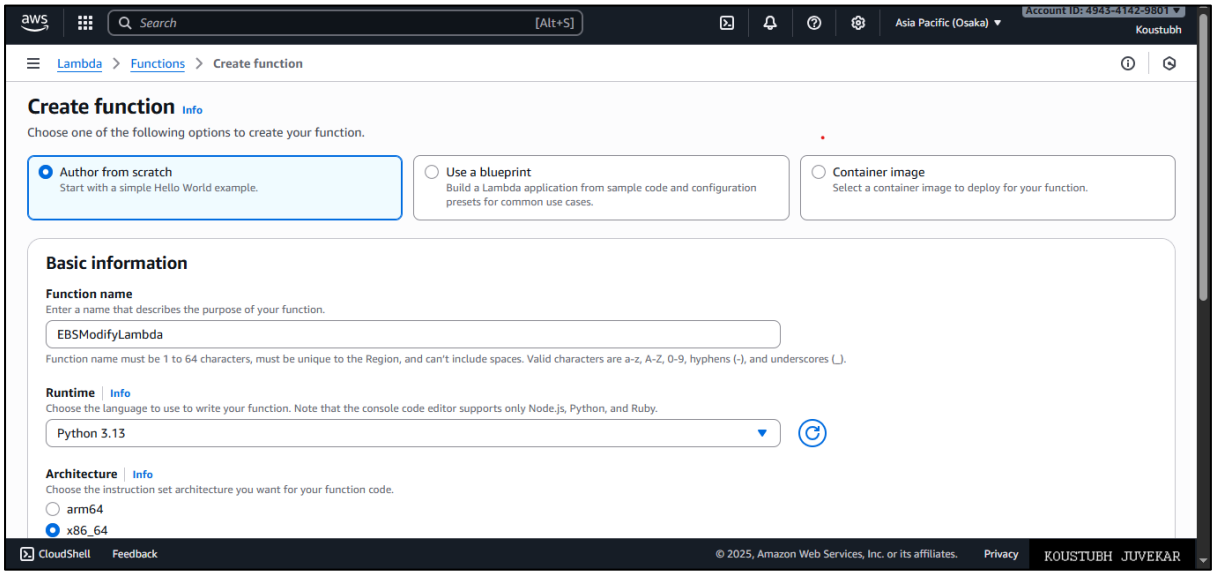

Image 6.6B : *Create Function - EBSModifyLambda.*

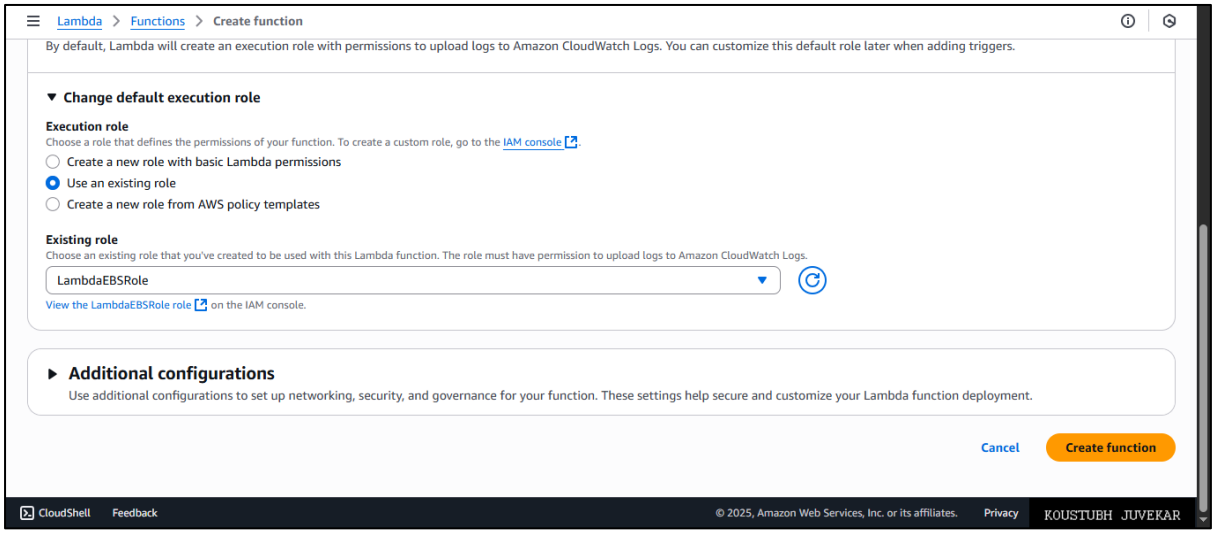

Image 6.7B : *Create Function - Existing role.*

- Add environment variable →
  - **Key -** DDB_TABLE
  - **Value -** EBSConversionLog
  - **Key -** SNS_TOPIC_ARN
  - **Value -** arn:aws:sns:ap-northeast-3:494341429801:EBSConversionTopic
- Click on **Save**

  It will be as
  - DDB_TABLE = EBSConversionLog
  - SNS_TOPIC_ARN = arn:aws:sns:ap-northeast-3:494341429801:EBSConversionTopic

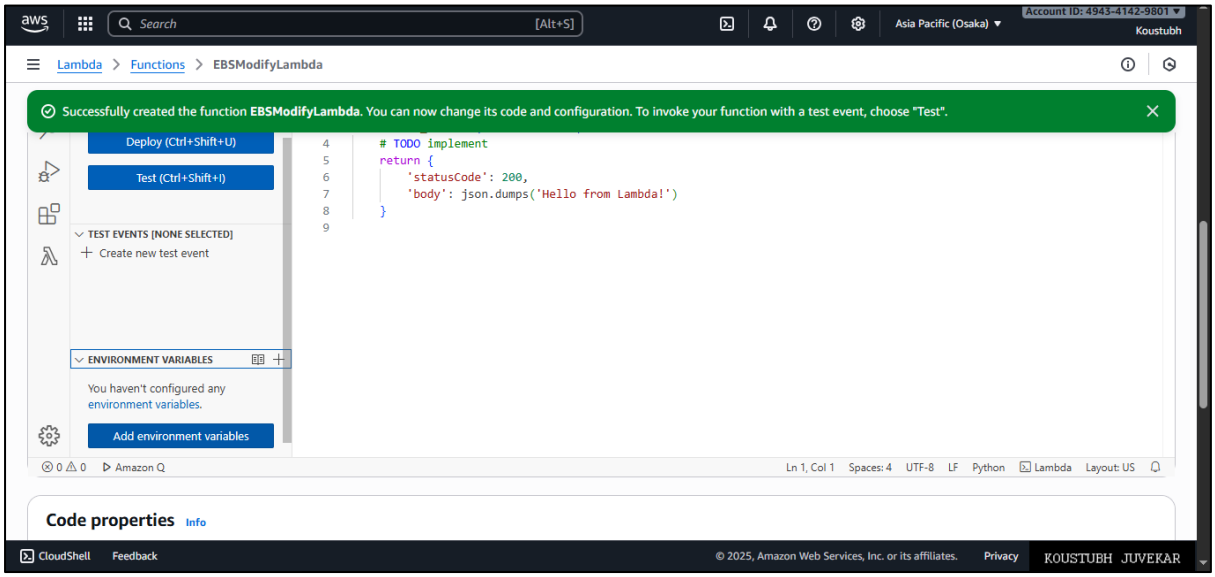

Image 6.8B : *Add environment variable.*

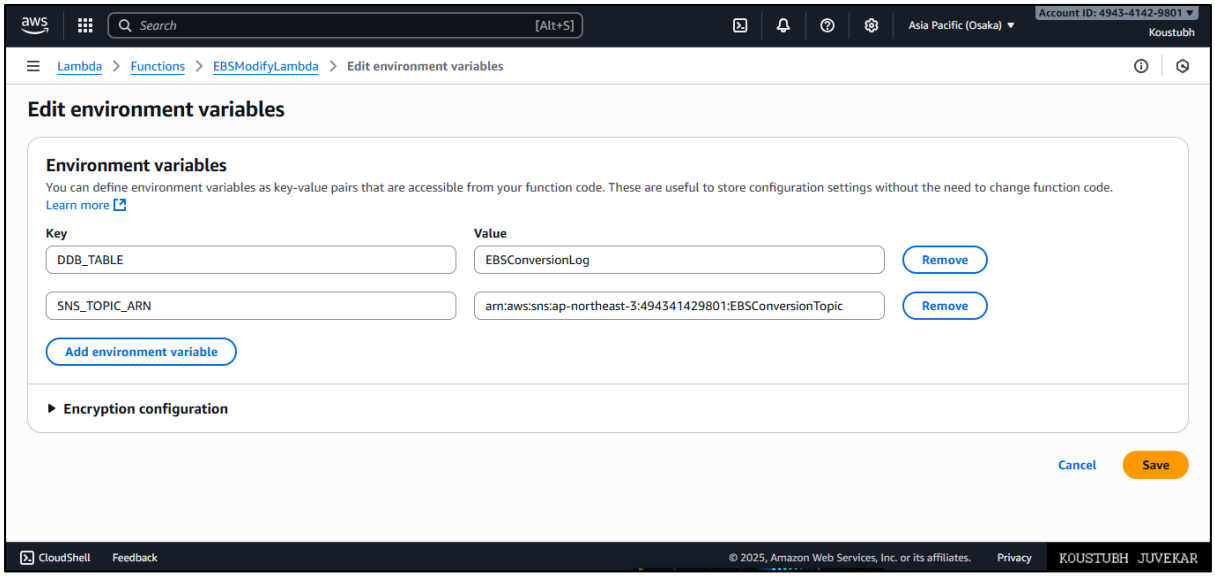

Image 6.9B : *Added environment variable.*

(P.T.O.)

- Paste code for modify function. *lambda_function.py*

```
import boto3
import json
import os
from datetime import datetime

def lambda_handler(event, context):
  print(f"EBS ModifyLambda started with event: {event}")

  ec2 = boto3.client('ec2')
  dynamodb = boto3.client('dynamodb')
  sns = boto3.client('sns')

  try:
      # Get volumes from Step Functions input
      volumes = event.get('Volumes', [])
      print(f"Processing {len(volumes)} volumes: {volumes}")

      if not volumes:
          return {
              'statusCode': 400,
              'body': json.dumps('No volumes provided in event')
          }

      converted_volumes = []

      for volume_id in volumes:
          print(f"Processing volume: {volume_id}")

          # Describe volume
          response = ec2.describe_volumes(VolumeIds=[volume_id])
          volume = response['Volumes'][0]

            # Check volume type
            current_type = volume['VolumeType']
            tags = {tag['Key']: tag['Value'] for tag in
volume.get('Tags', [])}

            print(f"Volume {volume_id}: type={current_type},
AutoConvert={tags.get('AutoConvert')}")

            if current_type == 'gp2' and tags.get('AutoConvert')
== '"true"':
                print(f"Converting volume {volume_id} from gp2 to
gp3")
```

```
                # Convert to gp3
                  ec2.modify_volume(VolumeId=volume_id,
VolumeType='gp3')

                # Update DynamoDB
                dynamodb.put_item(
                    TableName=os.environ['DDB_TABLE'],
                    Item={
                        'VolumeId': {'S': volume_id},
                          'Timestamp': {'S':
datetime.utcnow().isoformat()},
                        'Status': {'S': 'COMPLETED'},
                        'Action': {'S': 'Converted to gp3'}
                    }
                )

                # Send SNS notification
                try:
                    sns_response = sns.publish(
                        TopicArn=os.environ['SNS_TOPIC_ARN'],
                          Subject='✅ EBS Volume Converted
Successfully',
                          Message=f'''EBS Volume Conversion
Complete!

Volume ID: {volume_id}
Conversion: gp2 → gp3
Timestamp: {datetime.utcnow().isoformat()} UTC
Status: SUCCESS

Your EBS volume has been successfully converted from gp2 to gp3.
This may result in cost savings and improved performance.

AWS EBS Conversion Service'''
                    )
                        print(f"SNS notification sent:
 {sns_response['MessageId']}")
                except Exception as sns_error:
                    print(f"SNS notification failed:
 {str(sns_error)}")

                converted_volumes.append(volume_id)
                print(f"Successfully converted {volume_id} to
 gp3")
            else:
                print(f"Skipping {volume_id}:
 type={current_type}, AutoConvert={tags.get('AutoConvert')}")
```

```
      return {
          'statusCode': 200,
            'body': json.dumps(f"Converted
{len(converted_volumes)} volumes: {converted_volumes}")
        }

    except Exception as e:
        print(f"Error: {str(e)}")

        # Send error notification
        try:
            sns.publish(
                TopicArn=os.environ['SNS_TOPIC_ARN'],
                Subject='❌ EBS Volume Conversion Failed',
                Message=f'''EBS Volume Conversion Error!

Error: {str(e)}
Timestamp: {datetime.utcnow().isoformat()} UTC
Event: {json.dumps(event)}

Please check CloudWatch logs for more details.

AWS EBS Conversion Service'''
            )
        except:
            pass

        return {
            'statusCode': 500,
            'body': json.dumps(f"Error: {str(e)}")
        }
```

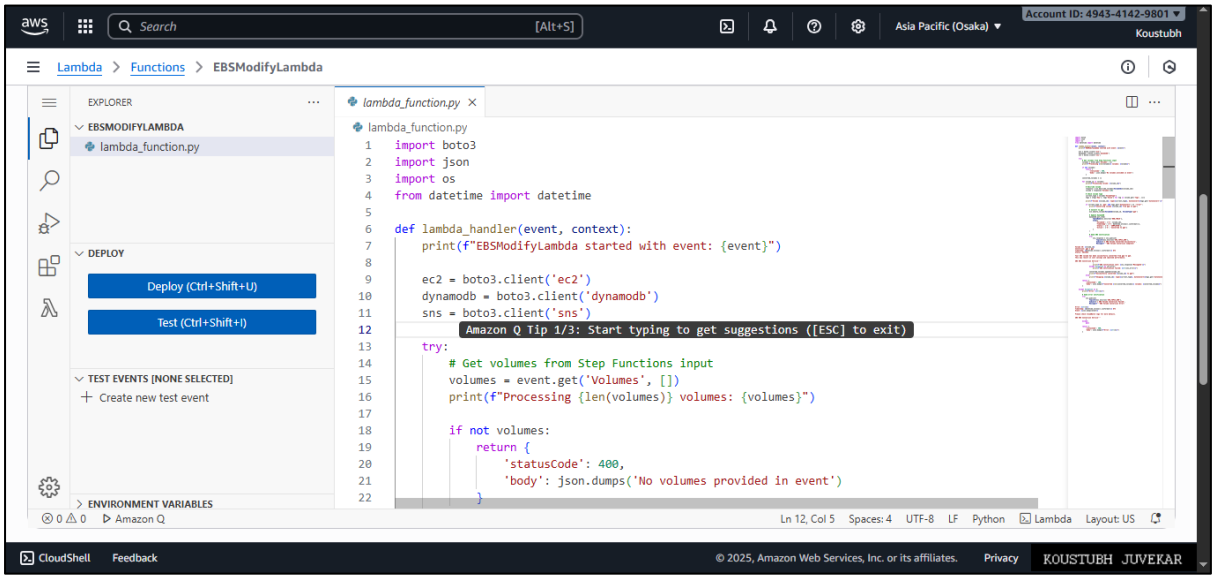

Image 6.10B : *Added code.*

- Go to **Configuration** → General Configuration → Click on Edit →
- Scroll down → Set **Timeout** - 5 min 0 sec
- Click on **Save.**

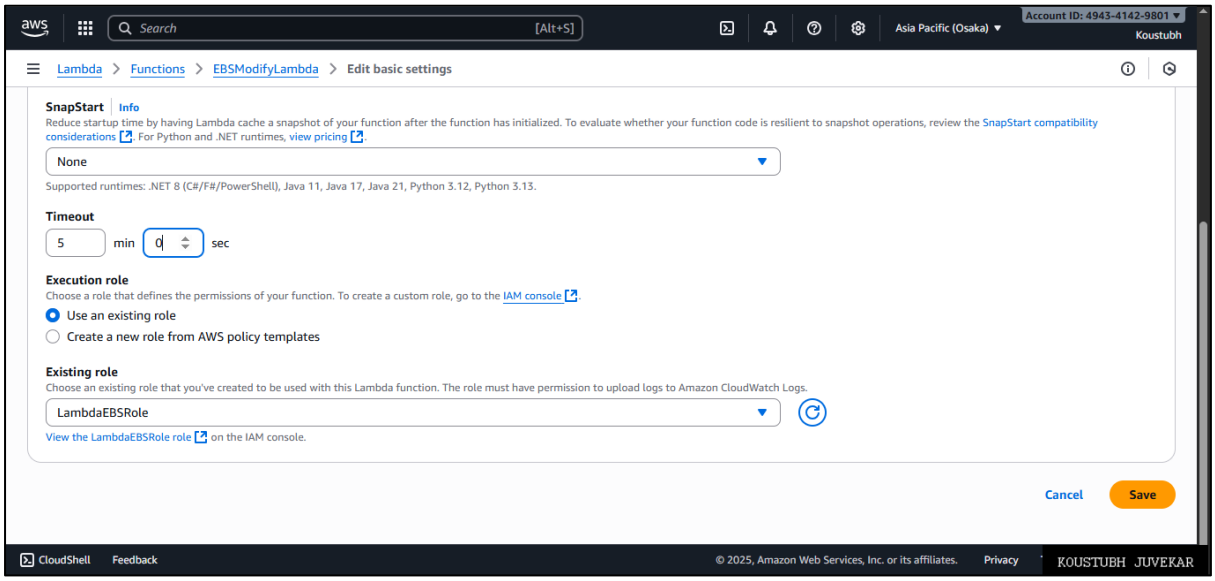

Image 6.11B : *Go to Configuration - Edit Timeout = 5min*

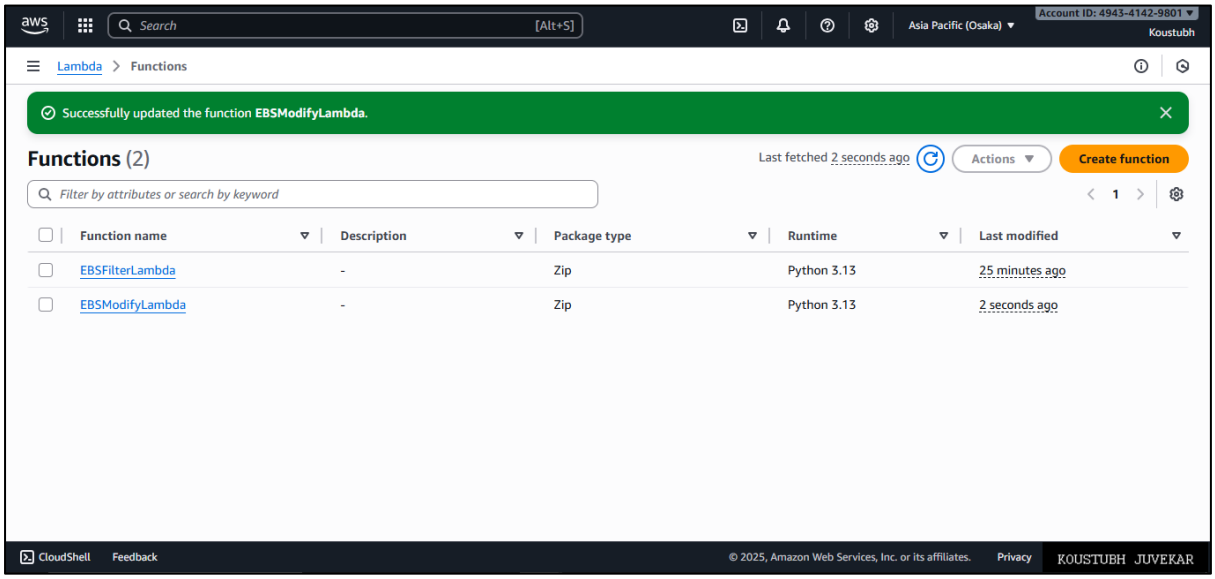

Image 6.12 : *Lambda Functions created*

(P.T.O.)

7. Create Step Function

- Go to **Step Functions → State Machines → Create State Machine**

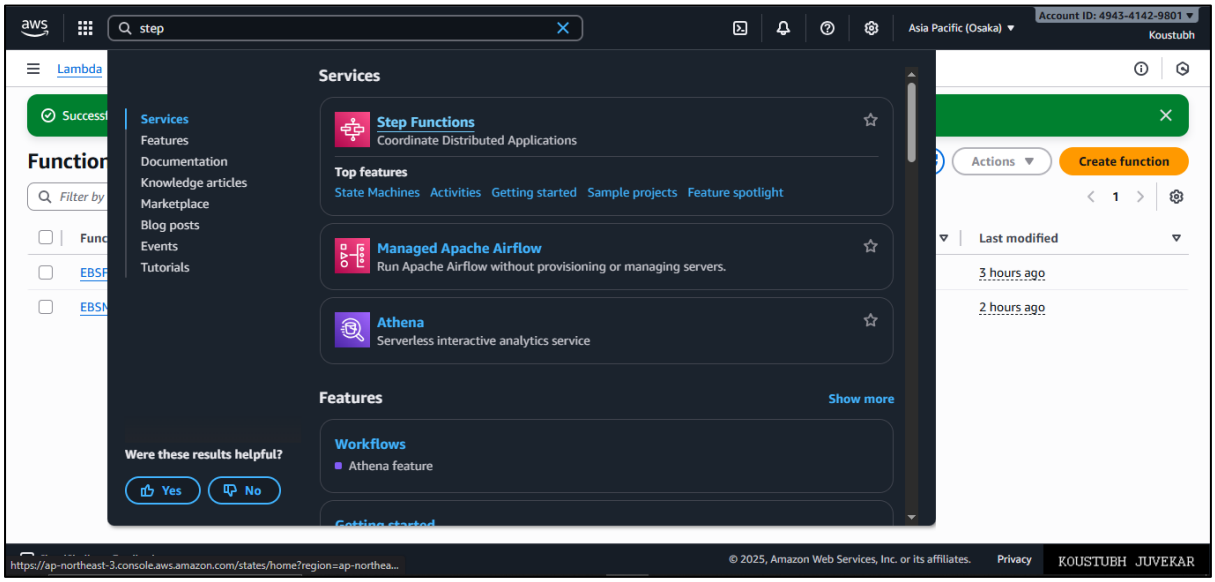

Image 7 : *Go to Step Function*

- Click on Create your own

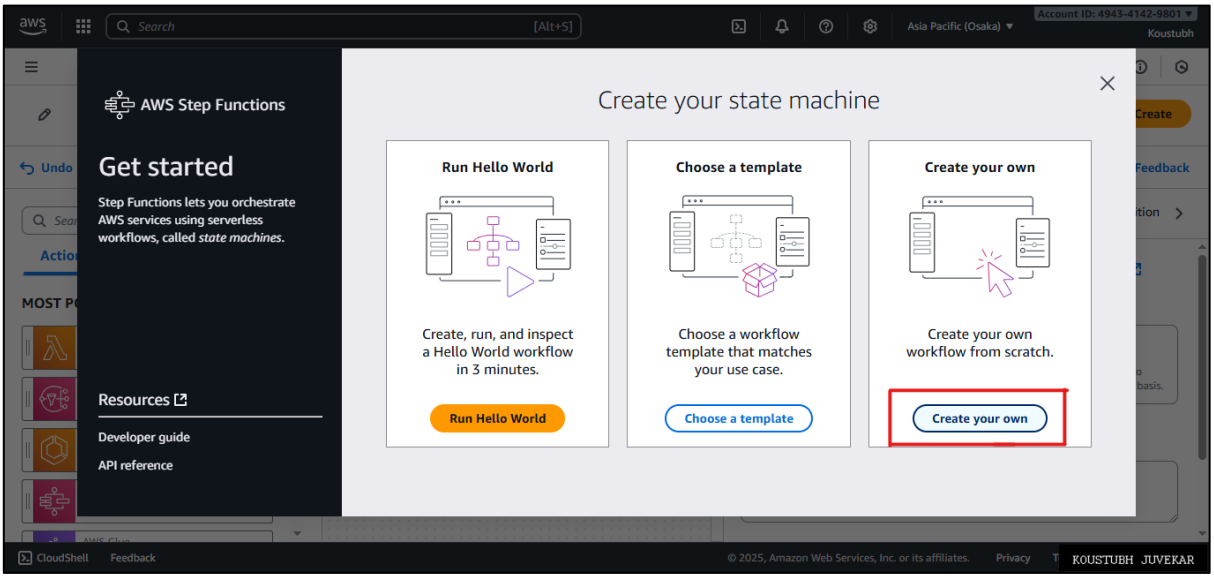

Image 7.1 *: Step Function - Create your own*

- Create State Machine
  - Step Machine name - **EBSConversionStateMachine**
  - Step Machine type - **Standard**
  - Click on **Continue**

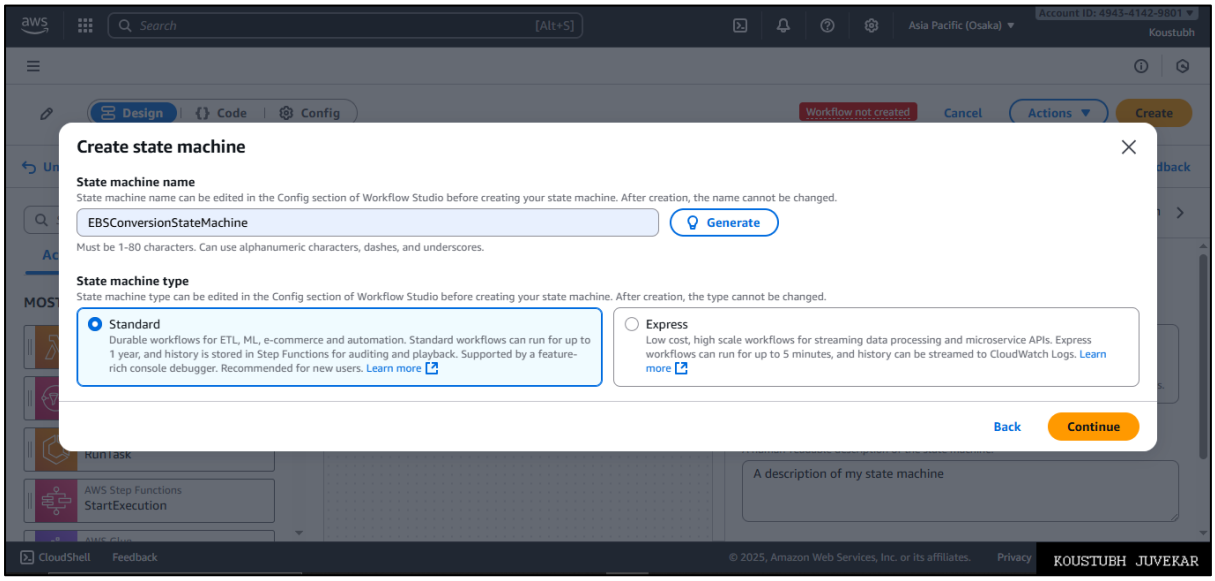

Image 7.1B : *Create state machine*

- Click on {} Code
- Paste JSON code here. (update ARNs). *EBSConversionStateMachine*

```
{
  "Comment": "State machine to convert gp2 volumes to gp3",
  "StartAt": "FilterVolumes",
  "States": {
    "FilterVolumes": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:ap-northeast-
3:494341429801:function:EBSFilterLambda",
      "ResultPath": "$.FilterResult",
      "Next": "CheckVolumesFound"
    },
    "CheckVolumesFound": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.FilterResult.Volumes",
          "IsPresent": true,
          "Next": "ModifyVolumes"
        }
      ],
      "Default": "NoVolumesFound"
    },
    "ModifyVolumes": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:ap-northeast-
3:494341429801:function:EBSModifyLambda",
```

```
        "InputPath": "$.FilterResult",
        "ResultPath": "$.ModifyResult",
        "End": true
      },
      "NoVolumesFound": {
        "Type": "Pass",
        "Result": "No gp2 volumes found with AutoConvert=true",
        "End": true
      }
    }
  }
```

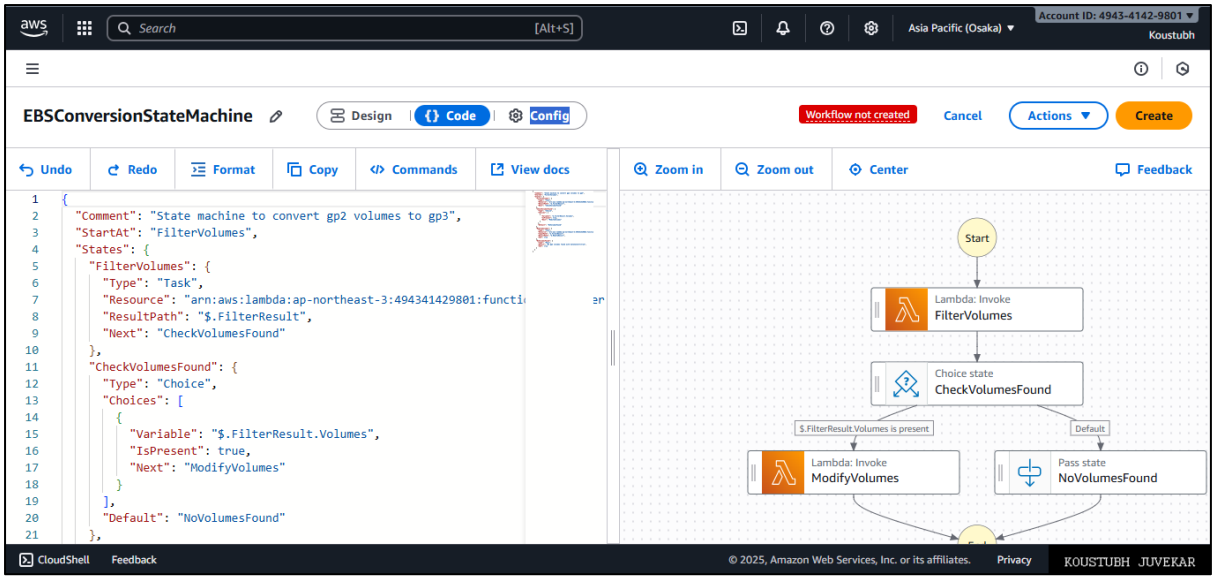

Image 7.2 : *Step Function - JSON Code*

- Click on {} Config → Permission →
  - Execution role →
    - Click on Drop down list → Choose and existing role → Select `StepFunctionsEBSRole`
- Click on **Create**

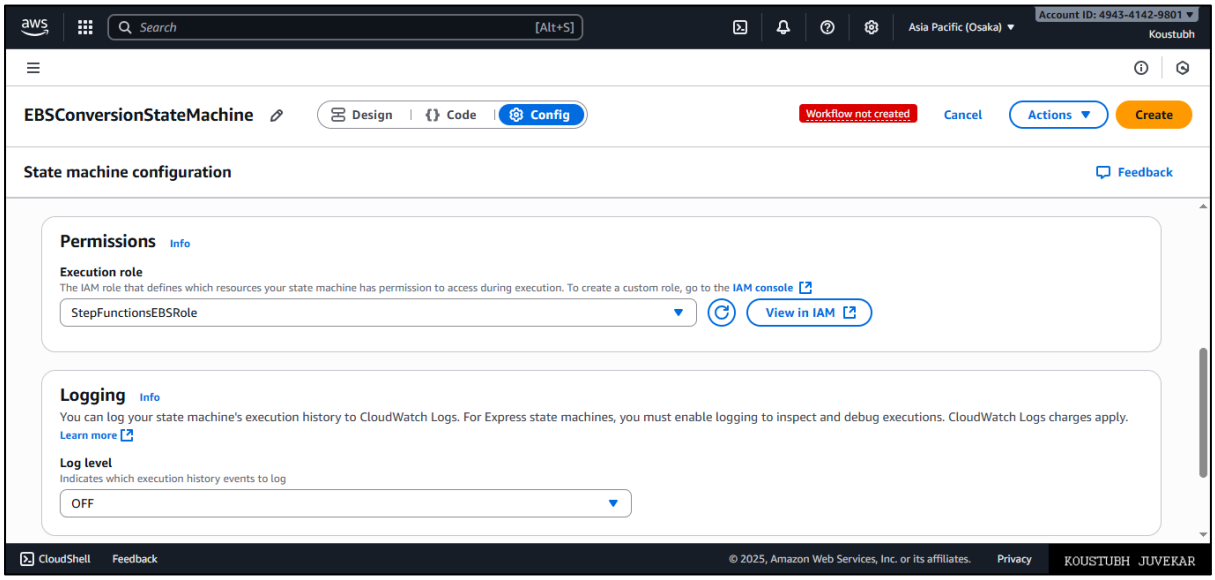

Image 7.3 : *Step Function - Configuration - Execution rule StepFunctionsEBSRole*

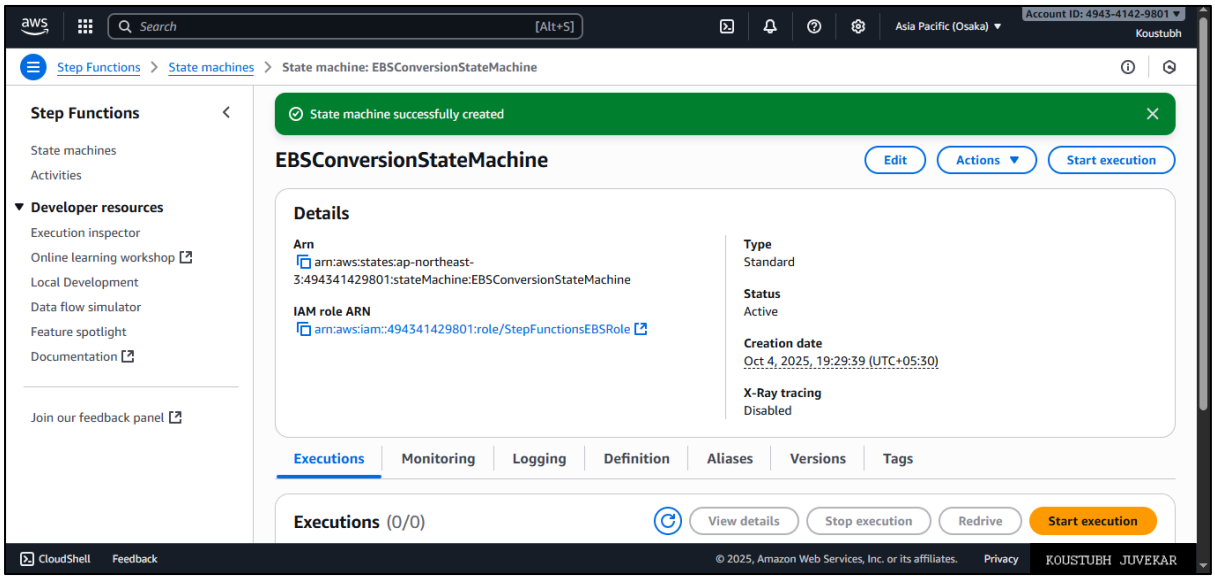

Image 7.4 : *Step Function created*

(P.T.O.)

8. Create EventBridge Rule

- Go to **EventBridge → Rules →** Create Rule

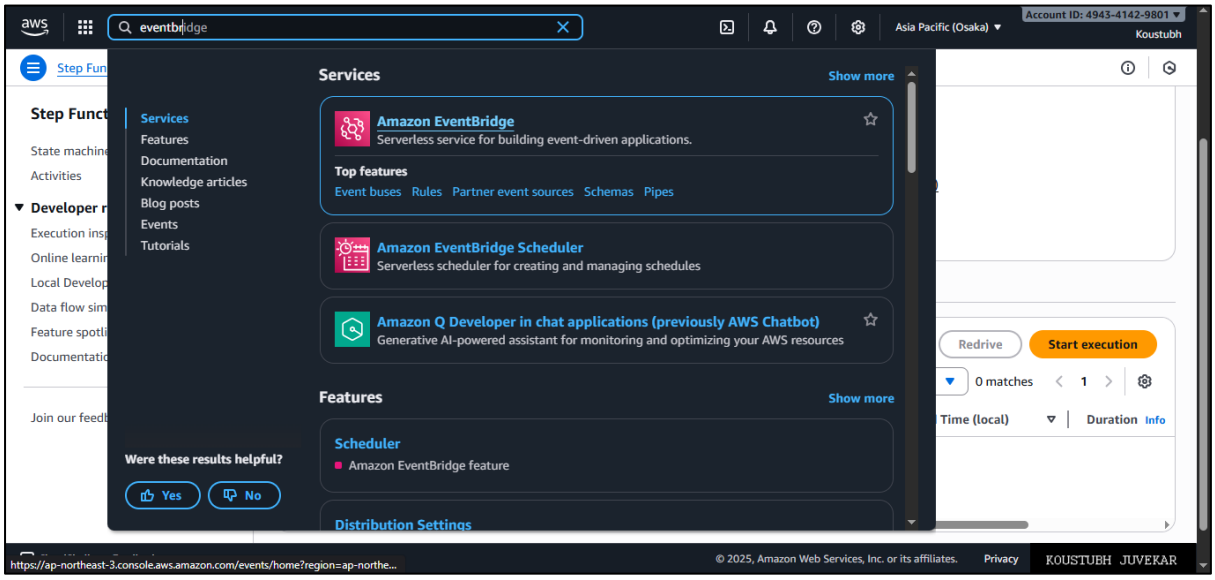

Image 8 : *Go to Eventbridge console*

- Define rule detail → Rule detail →
  - **Name -** EBSConversionDaily
  - **Event bus -** default
  - **Rule type -** Schedule
- Click on **Continue to create rule**

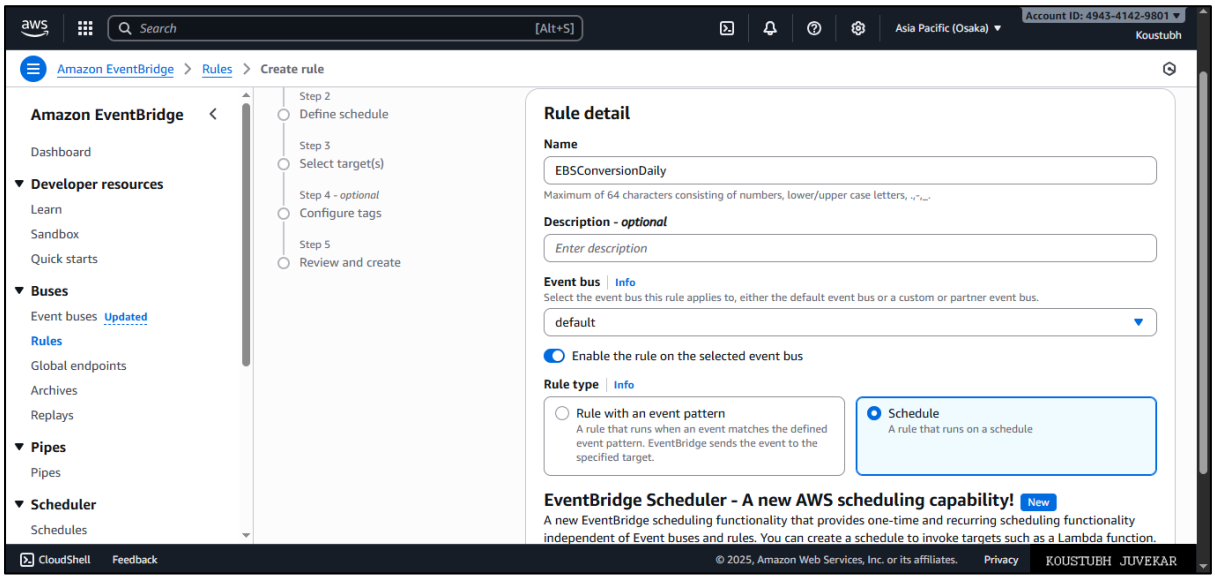

Image 8.1 : *Eventbridge console - rule 1*

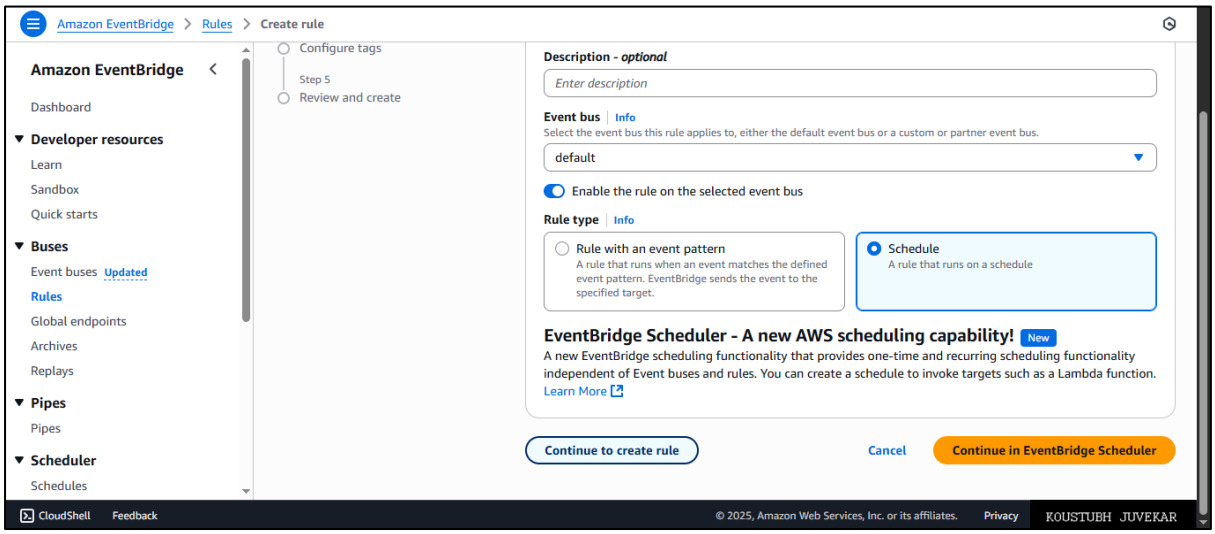

Image 8.2 : *Continue to create rule*

- Define schedule → Schedule pattern →

  A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.

- Cron Expression
  - cron(0 2 * * ? *) (runs daily at 2AM UTC). *(Set time as per your requirement. If you want immidiate testing, set time accordingly.)*
- Next 10 triggers dates and timings will be displayed.
- Click on Next
- Select target(s) → Target 1 → Select AWS service
  - **Select a target** - Step Functions state machine
  - **State machine** - EBSConversionStateMachine
  - **Execution role** - Create a new role for this specific resource
  - **Role name** - EventBridgeInvokeStepFunctionRole1
  - **Click on** **Next**
- Keep remaining options as it is. At the end, EventBridge Rule will be created.

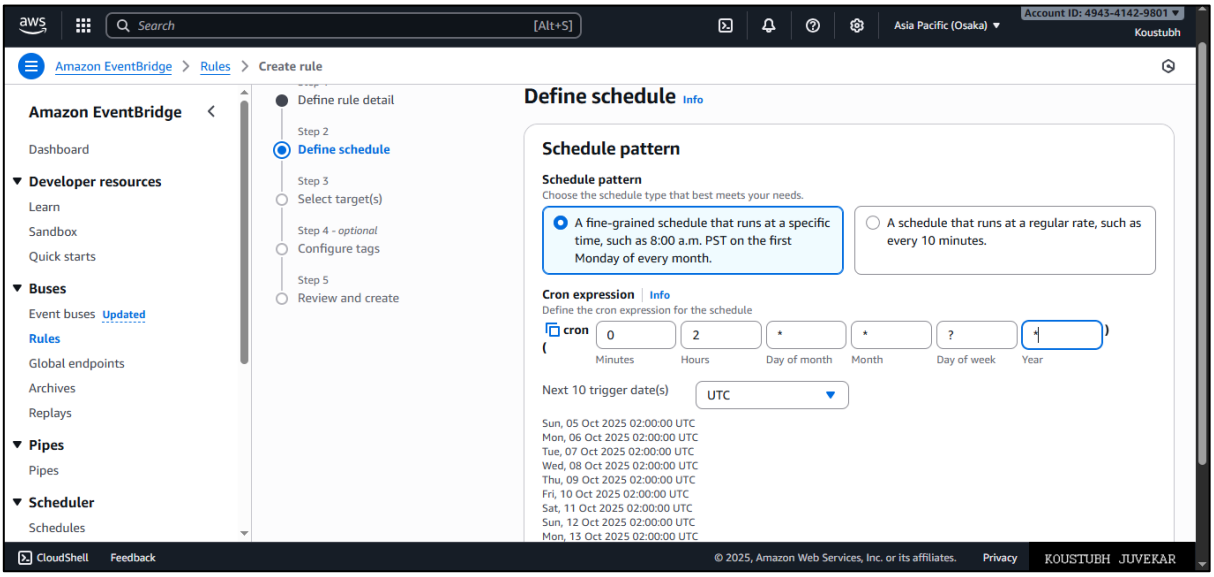

Image 8.3 : *Continue to Create schedule*

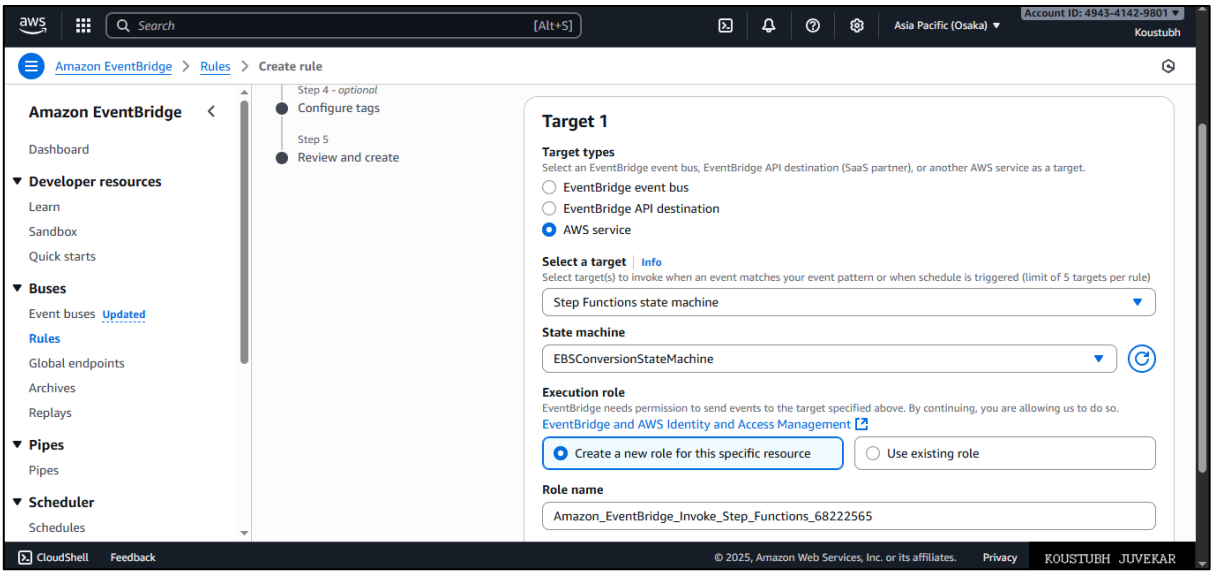

Image 8.4 : *Continue to Set Target*

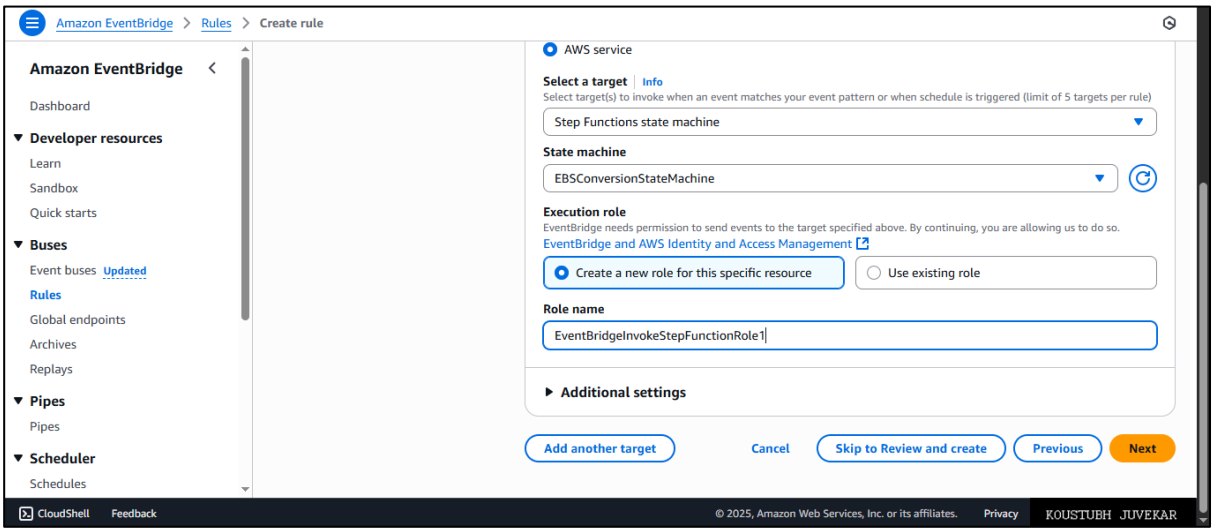

Image 8.5 : *Continue to create IAM role EventBridgeInvokeStepFunctionRole1*

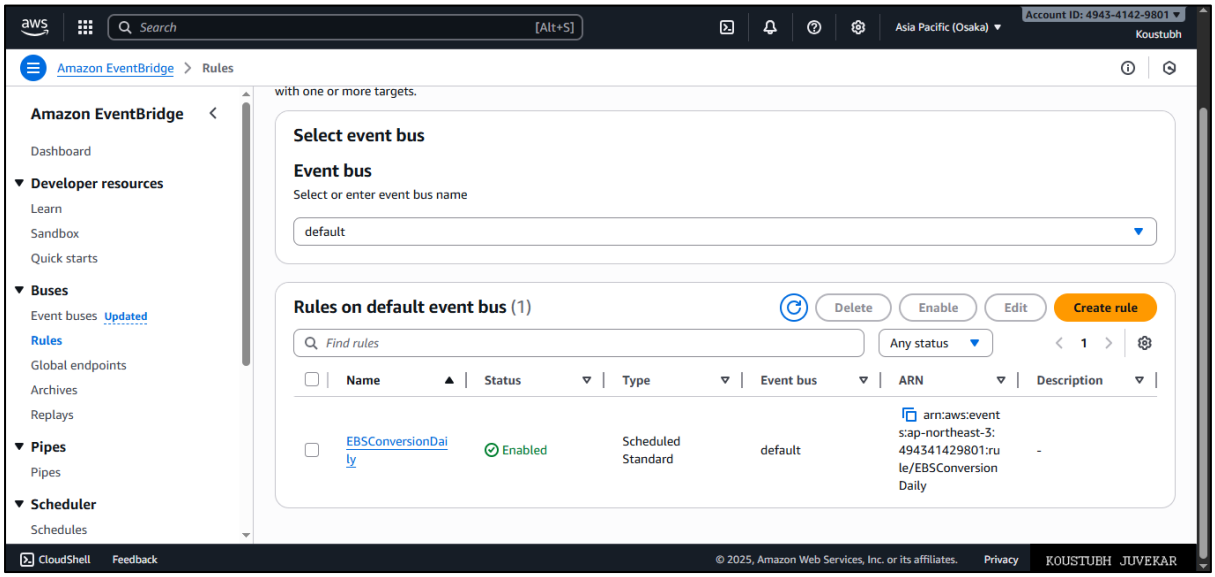

Image 8.6 : *Eventbridge rule created*

9. Testing & Validation

- Before test -

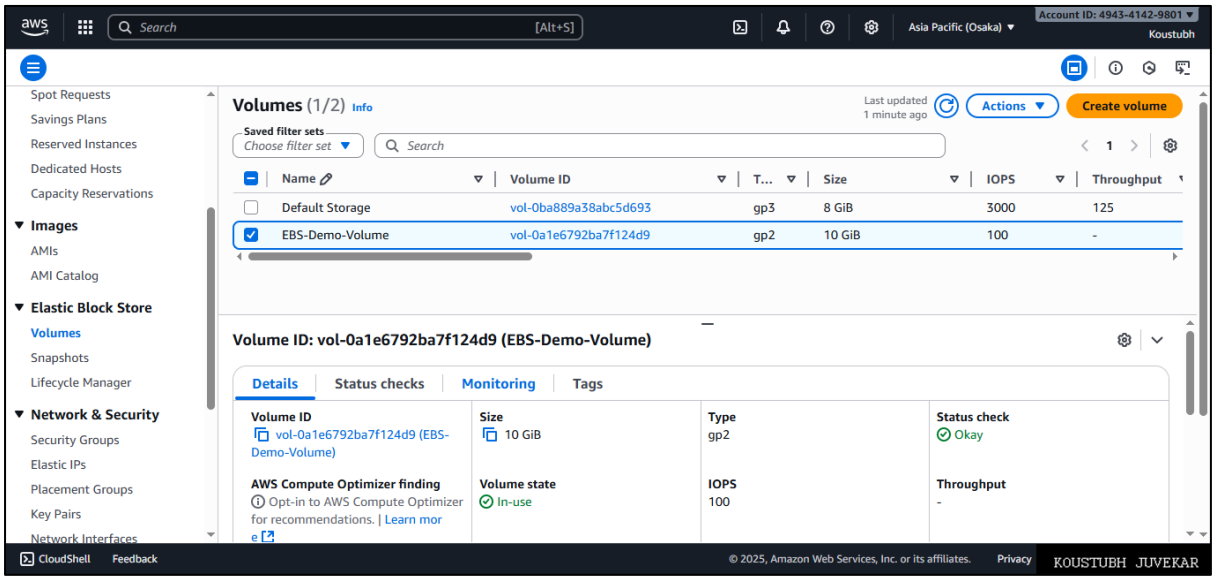

Image 9 : *Screenshot before testing*

- Manually start **Step Function** execution:
  - Go to Step Functions → State machines → State machine:
    `EBSConversionStateMachine`
  - Click on **Start Execution**

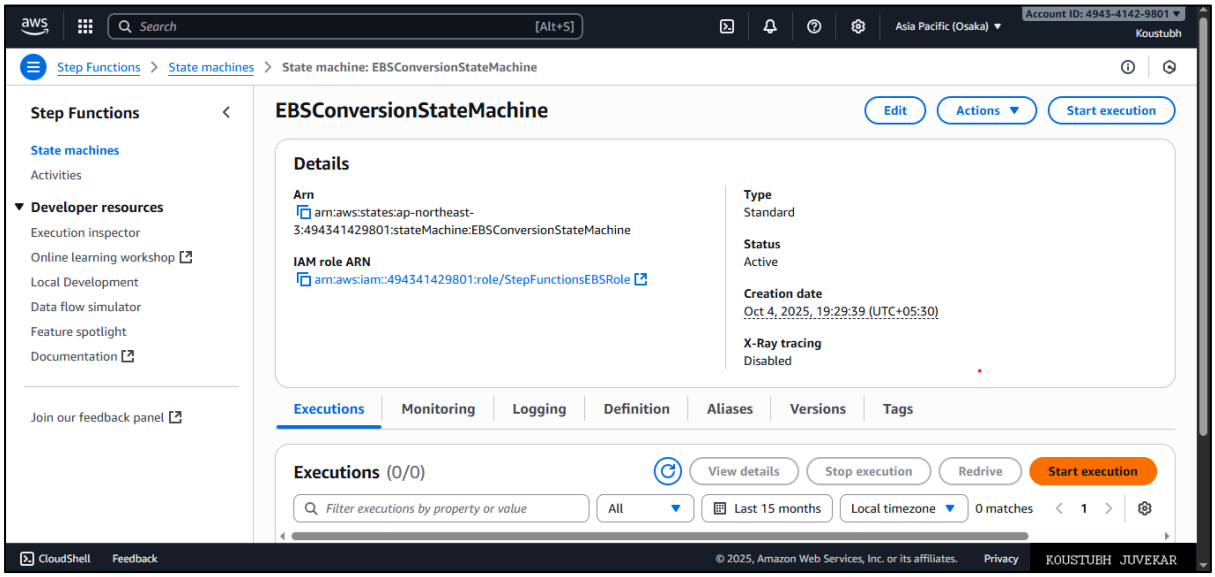

Image 9.1 : *Step function - Execution start*

- Input: {}
- Click on **Start execution**
- Execution will be started. Wait until success.

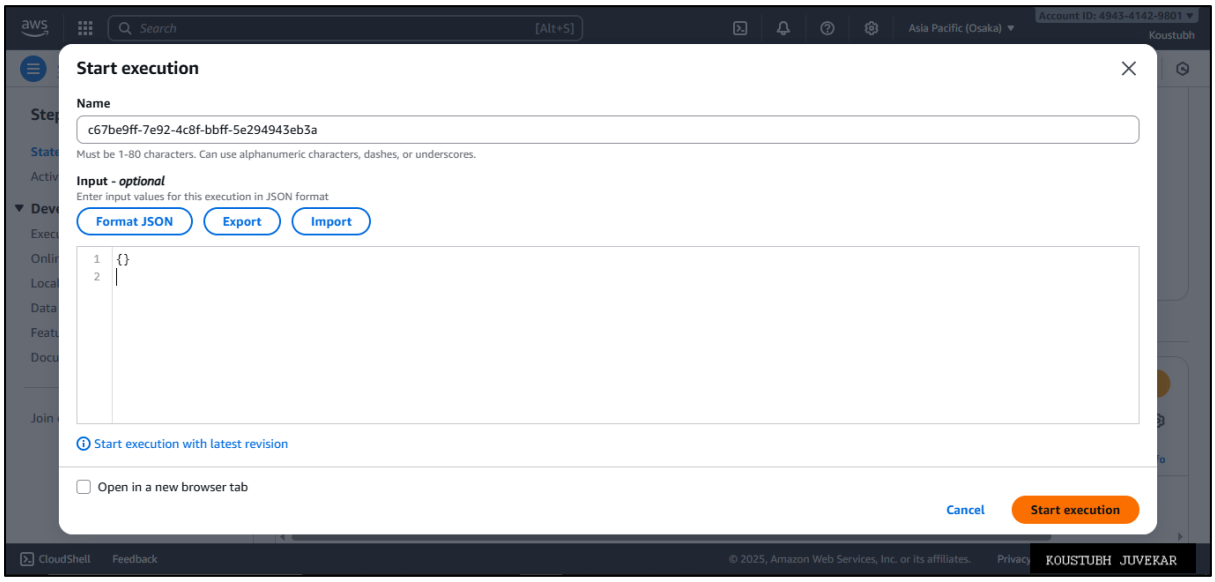

Image 9.2 : *Step function - Execution start - Input*

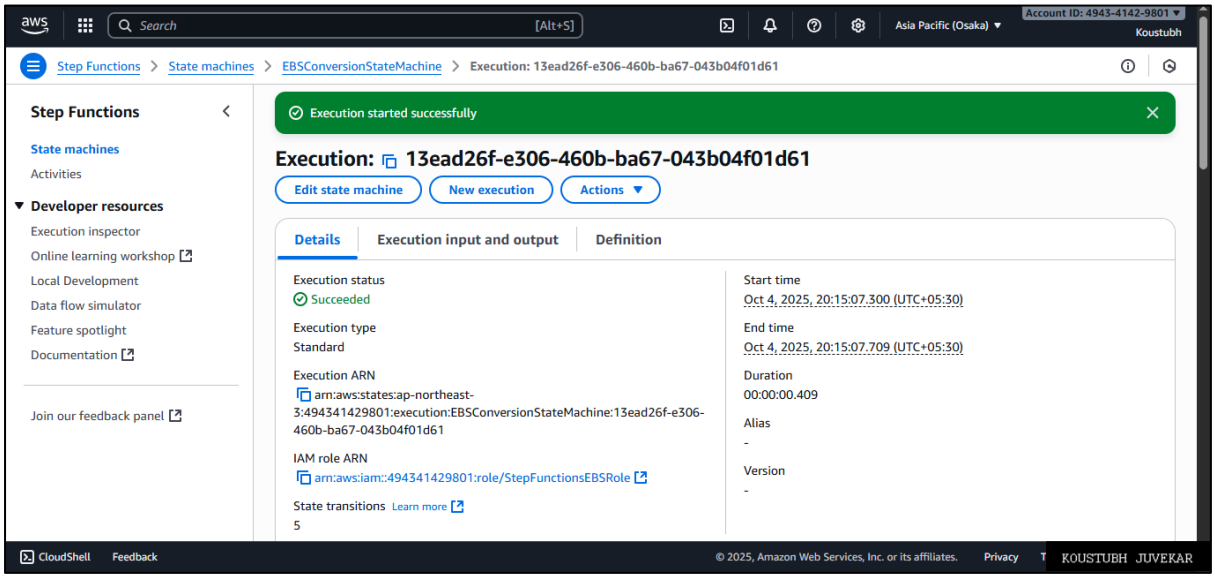

Image 9.3 : *Step function - Execution start - Execution succeeded*

- After succession, graph view of execution will be displayed.

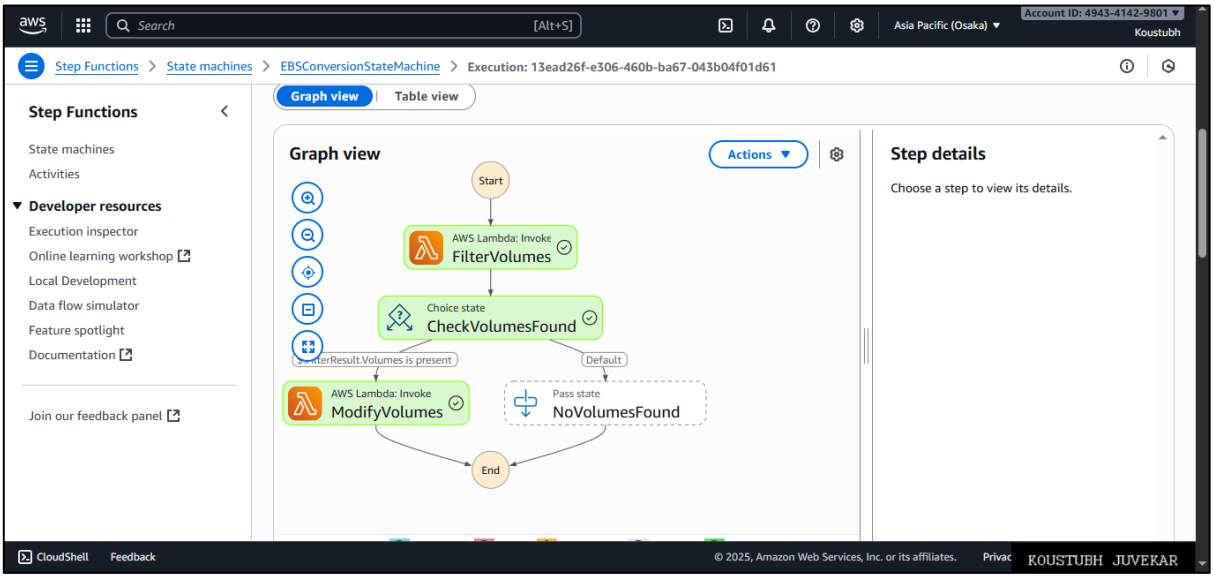

Image 9.4 : *Graph view*

- After run:
  - Go to **DynamoDB → Items tab → Explore Items → Check for DynamoDB audit log** → *should see entry.*

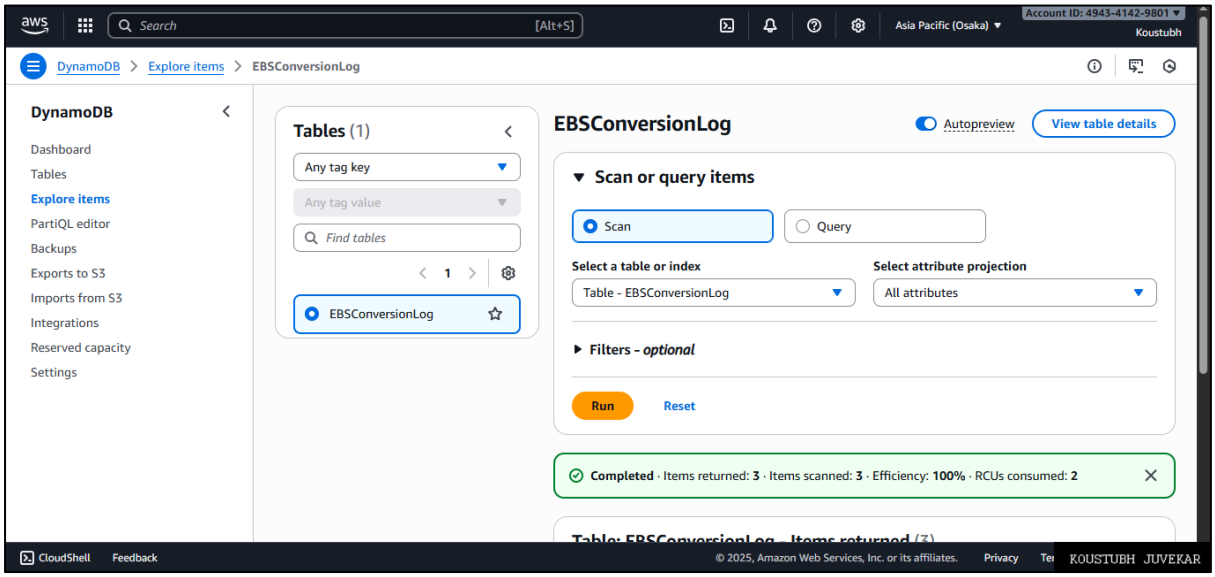

Image 9.5 : *DynamoDB audit log*

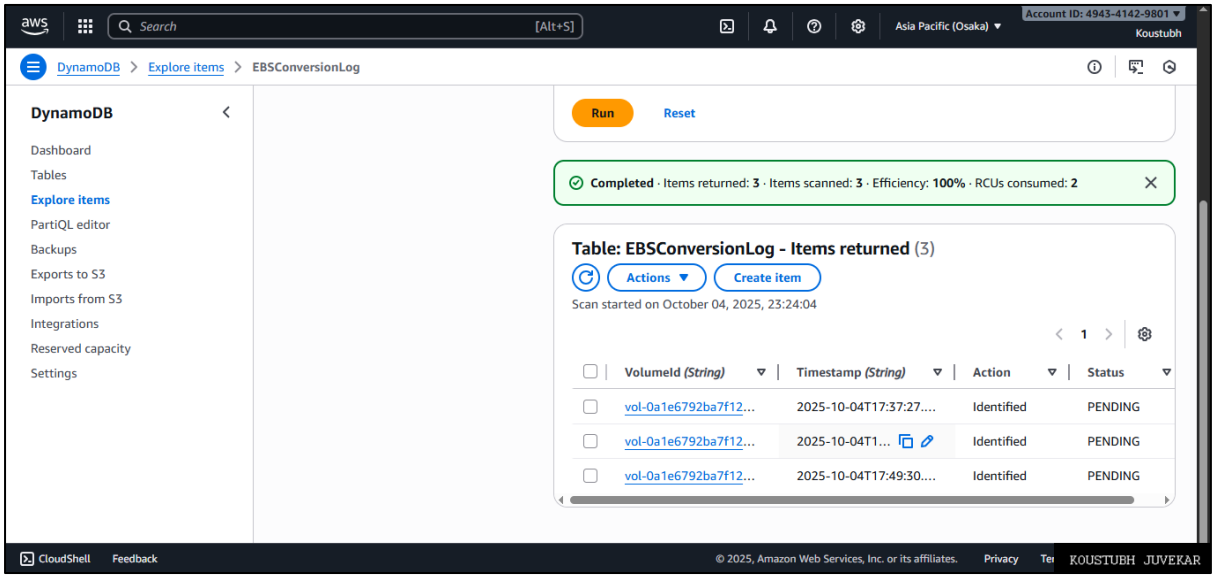

Image 9.6 : *Verify DynamoDB audit log*

- To check execution step by step and CloudWatch logs showing Lambda output -
  - Go to **CloudWatch → Log groups → /aws/lambda/EBSFilterLambda**

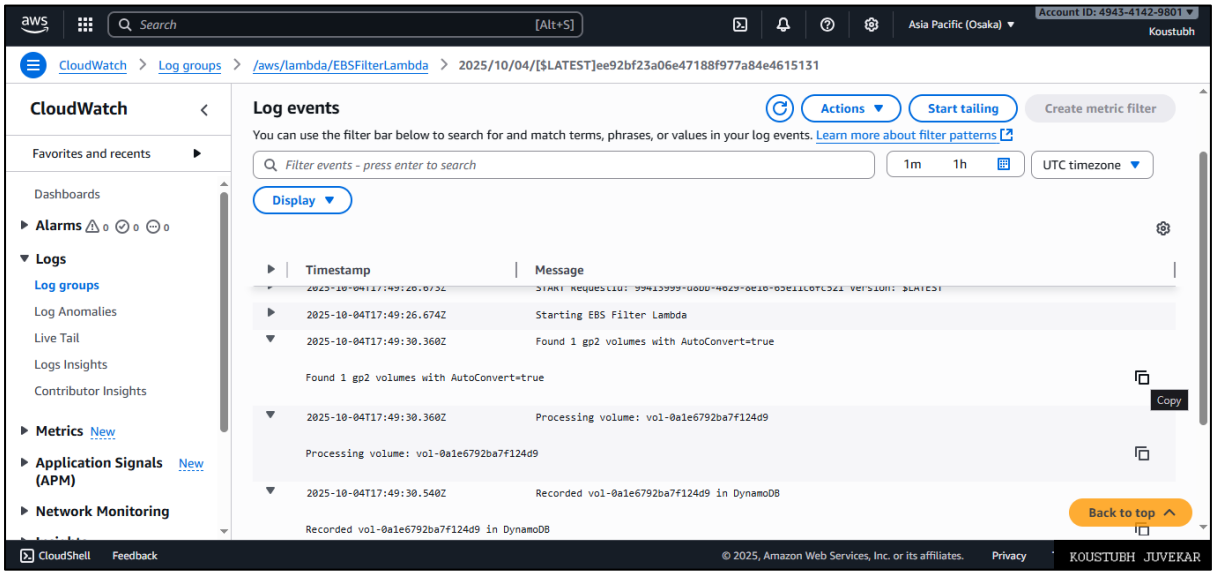

Image 9.7A : *CloudWatch → EBSFilterLambda → Converting gp2 to gp3*

- Go to **CloudWatch → Log groups → /aws/lambda/EBSModifyLambda**

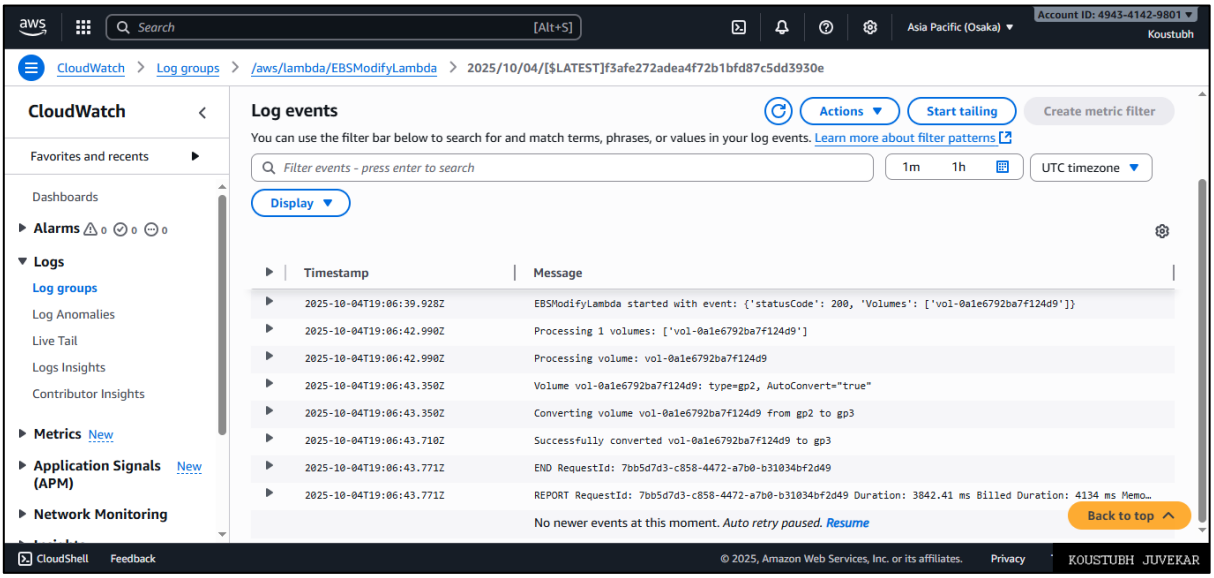

Image 9.7B : *CloudWatch → EBSModifyLambda → Converted gp2 to gp3*

- Step Function execution process → Here, scroll down and check for step column.
- It will show ModifyVolumes Status

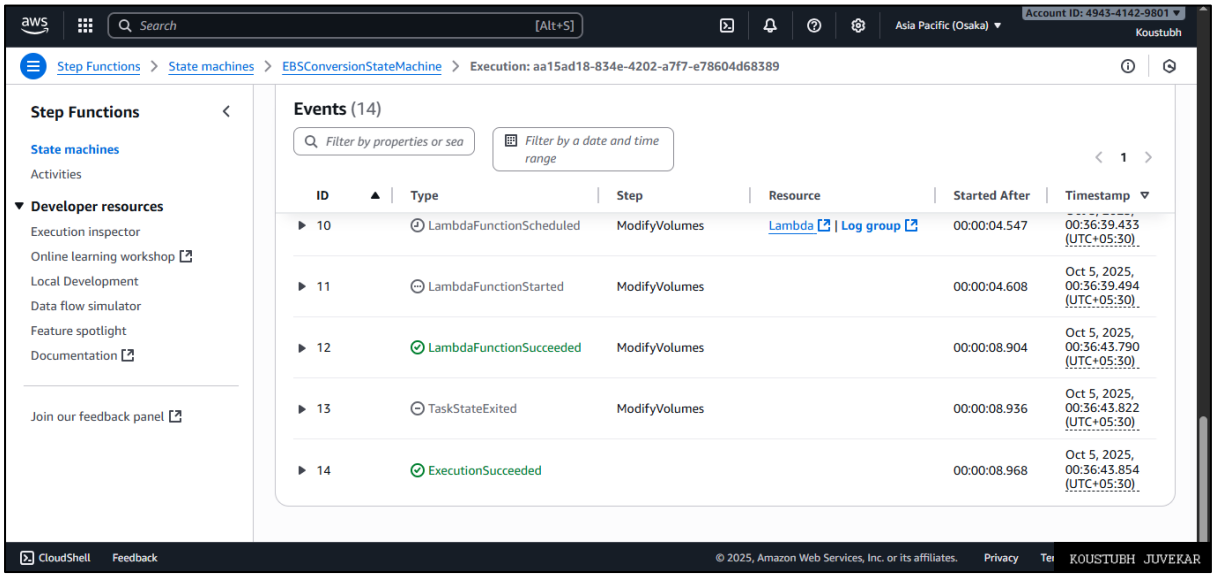

Image 9.8 : *Converted gp2 to gp3 - step function execution*

- Now go to EC2 console → Elastic Block Store → Volumes
- Check for volume which was gp2 initially, now it is gp3, after conversion.

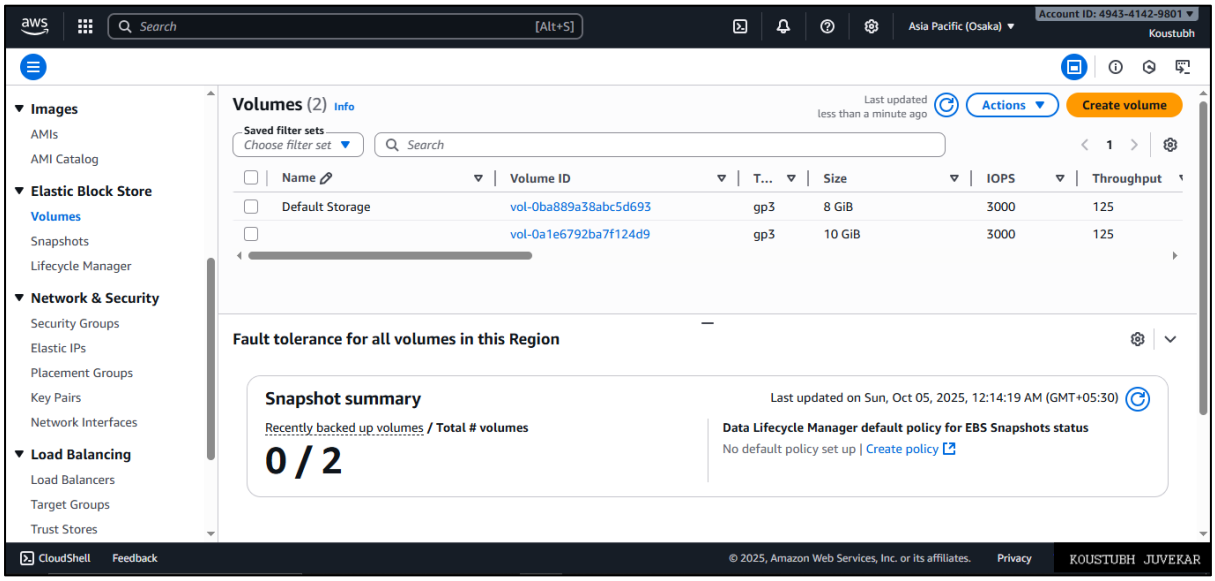

Image 9.9 : *Converted gp2 to gp3 Volume*

- Created another gp2 volume for testing and attached to EC2 → `vol-005370e741329e7d3`
- After Step Function Execution and complete conversion of volume, received email update of service SNS

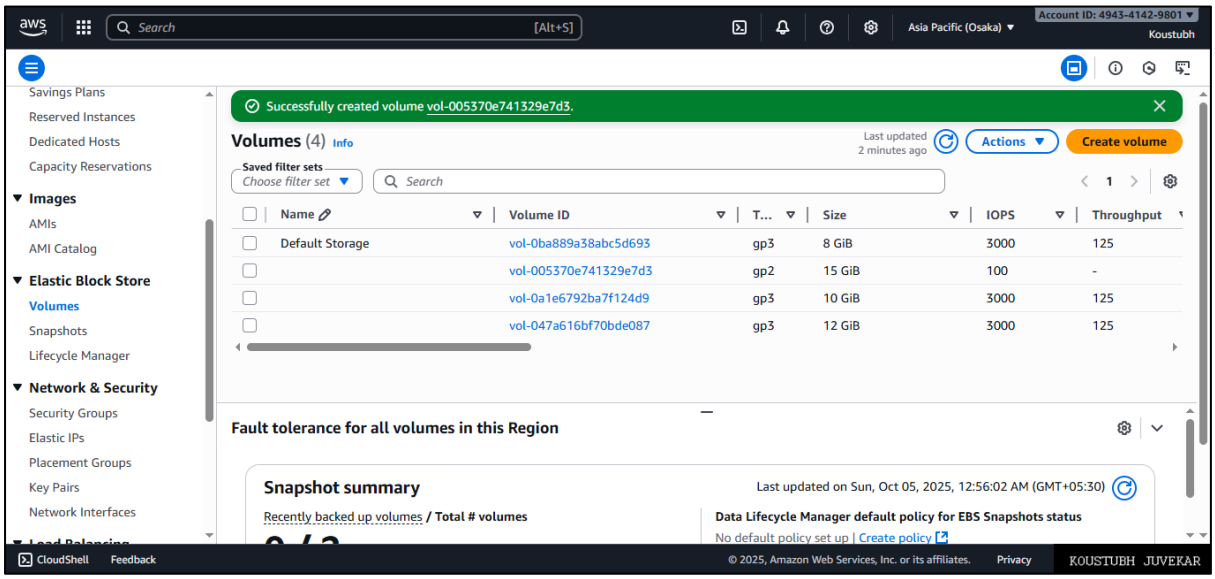

Image 9.10 : *Converted gp2 to gp3 Volume page - created another gp2 - vol-005370e741329e7d3*

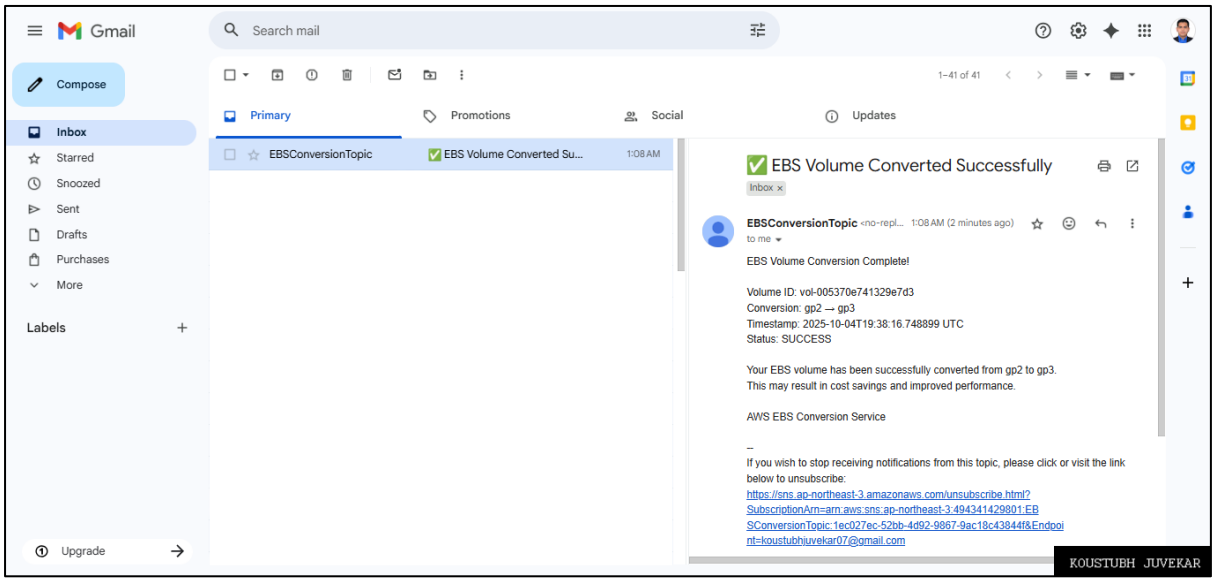

Image 9.11 : *Converted gp2 to gp3 Volume page - created another gp2 to check SNS - recieve email for vol-005370e741329e7d3*

- DynamoDB logs

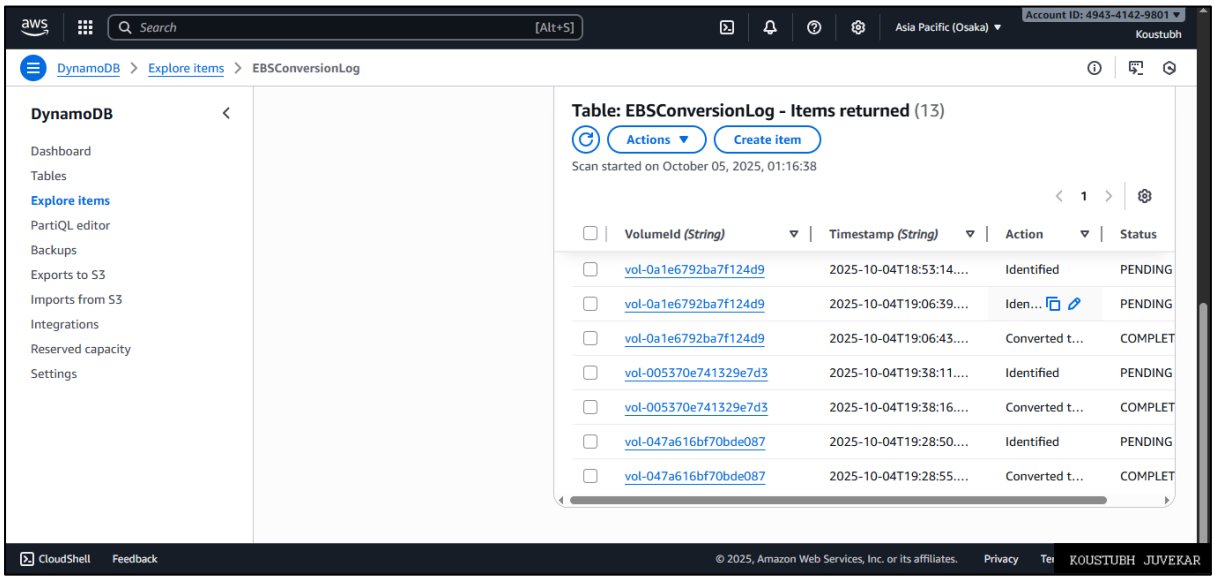

Image 9.12 : *Converted gp2 to gp3 Volume page - DynamoDB logs*

- Go to Volumes → All volumes are automatically converted to gp3.

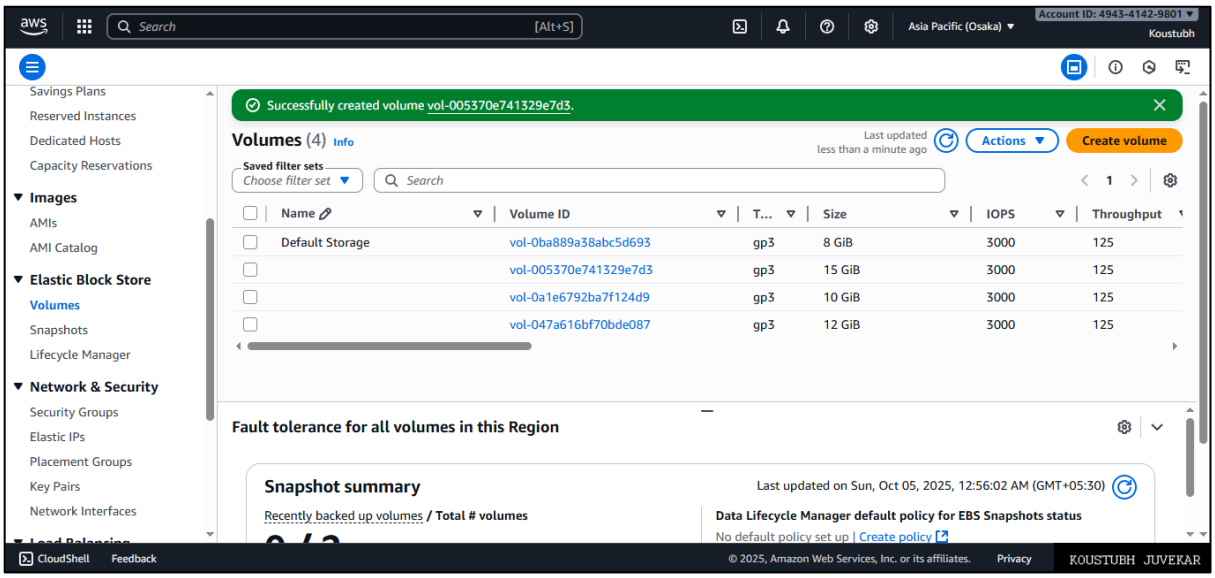

Image 9.13 : *Converted all gp2 to gp3*

(P.T.O.)

- Cloudwatch log groups

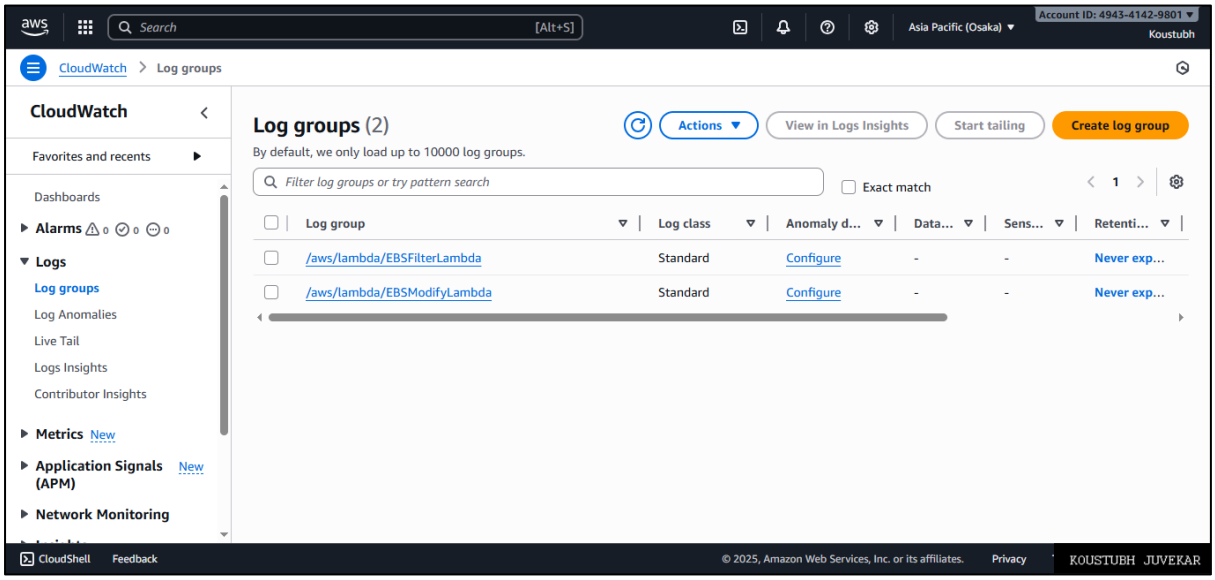

Image 9.14 : *Cloudwatch for both groups*

- Cloudwatch log Events for EBSModifyLambda

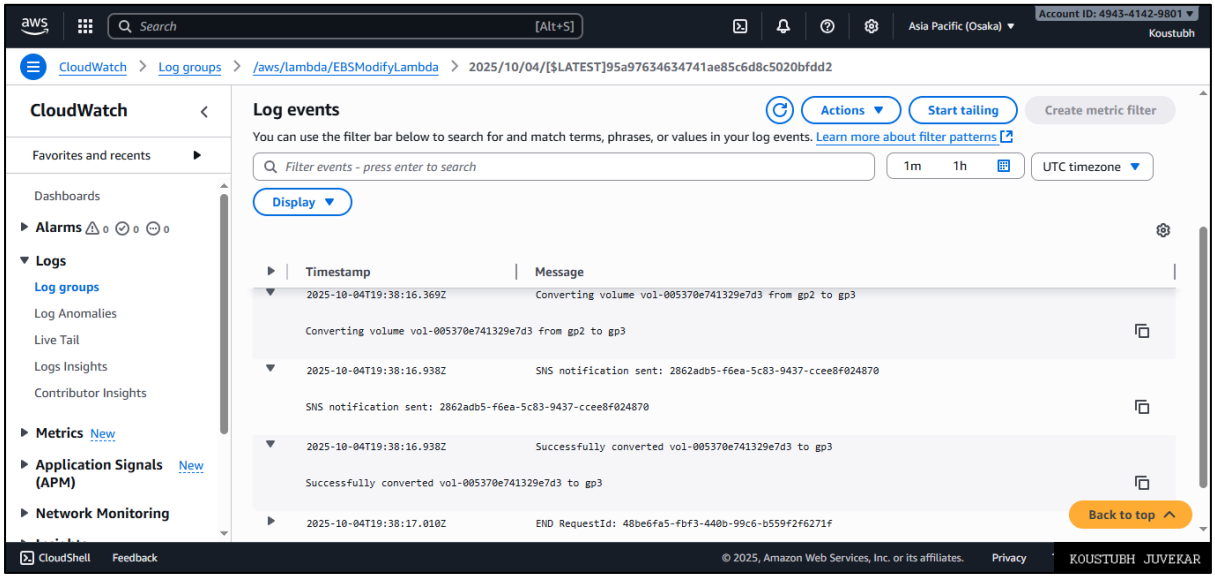

Image 9.15 : *Cloudwatch log Events for EBSModifyLambda*

(P.T.O.)

- Cloudwatch log Events for EBSFilterLambda

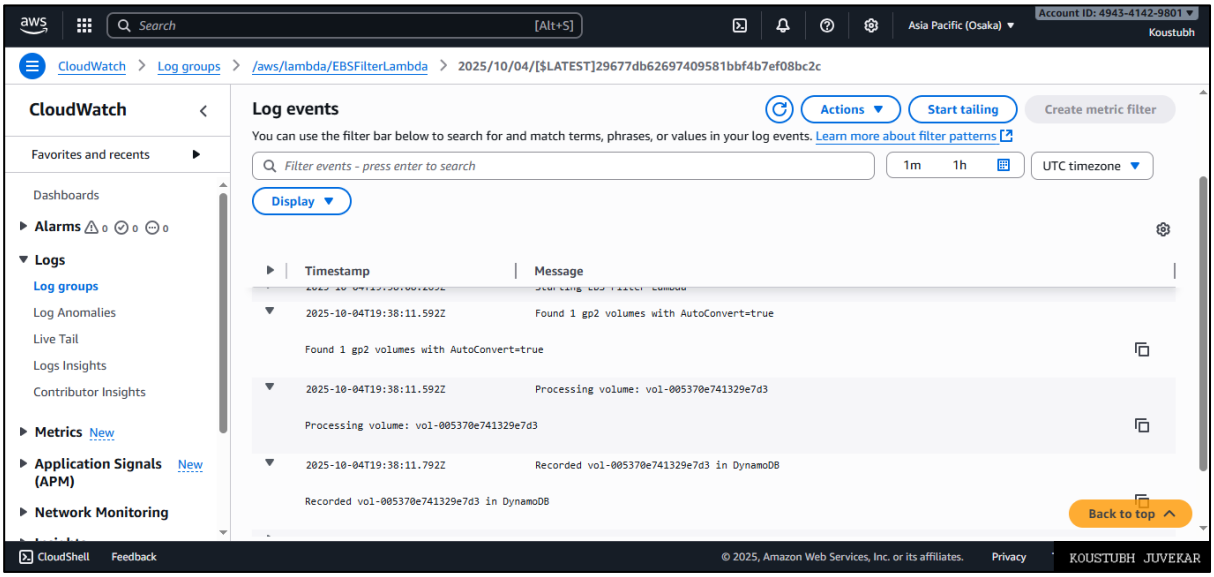

Image 9.15 : *Cloudwatch log Events for EBSFilterLambda*

(P.T.O.)

❖ **Architecture diagram of your workflow**

Intelligent EBS Volume Optimization Pipeline

```
│
├── EventBridge (Scheduled Trigger)
│      └── Triggers Step Function Daily
│
├── Step Functions: EBSConversionStateMachine
│      │
│      ├── Step 1: FilterVolumes (Lambda: EBSFilterLambda)
│      │       ├── Scans all EBS volumes
│      │       ├── Filters: VolumeType = gp2
│      │       ├── Filters: Tag AutoConvert = true
│      │       └── Logs "Identified" items into DynamoDB
│      │
│      ├── Choice State: CheckVolumesFound
│      │       ├── If volumes found → ModifyVolumes
│      │       └── If none → End workflow
│      │
│      └── Step 2: ModifyVolumes (Lambda: EBSModifyLambda)
│              ├── Converts gp2 → gp3 using modify_volume()
│              ├── Updates DynamoDB: Status=COMPLETED
│              ├── Publishes SNS Email Notification
│              └── Sends SUCCESS/ERROR message
│
├── DynamoDB (Audit Logs)
│      ├── Table: EBSConversionLog
│      ├── Records:
│      │       ├── VolumeId
│      │       ├── Timestamp
│      │       ├── Status (PENDING / COMPLETED)
│      │       └── Action (Identified / Converted)
│
```

```
├── SNS Topic
│     ├── Sends email notifications for:
│     │       ├── Successful volume conversions
│     │       └── Error/failure alerts
│
└── CloudWatch Logs
      ├── Logs for EBSFilterLambda
      ├── Logs for EBSModifyLambda
      └── Step Functions execution history
```

❖ **Any real-world challenges simulated and how handled**

Challenge 1: Volume Conversion Delay

Problem:
EBS gp2 → gp3 conversion is asynchronous. It may take a few seconds to minutes.

Solution:
Used Step Functions + optional Wait state in real scenarios to confirm conversion via: describe-volume-modifications.

Challenge 2: Lambda Timeouts

Problem:
Volume conversion may exceed default Lambda timeout (3 seconds).

Solution:
Increased Lambda timeout to **5 minutes** in configuration.

Challenge 3: IAM Permissions Errors

Problem:
Lambda initially failed to modify volume due to insufficient permissions.

Solution:
Attached correct policies:

- AmazonEC2FullAccess
- AmazonDynamoDBFullAccess
- AmazonSNSFullAccess
- CloudWatchLogsFullAccess

- - END OF DOCUMENT - -