

Graded Assignment On Serverless Architecture

Git Repository link : https://github.com/igSpanser/Assignment_Serverless_Architecture.git

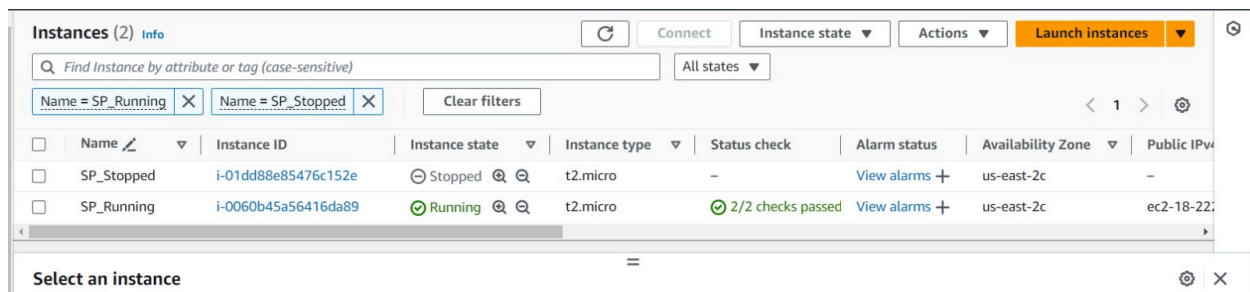
Assignment 1:

Automated Instance Management Using AWS Lambda and Boto3.

In this assignment, you will gain hands-on experience with AWS Lambda and Boto3, Amazon's SDK for Python. You will create a Lambda function that will automatically manage EC2 instances based on their tags.

1. EC2 Setup:

- Navigate to the EC2 dashboard and create two new t2.micro instances (or any other available free-tier type).
- Tag the first instance with a key `Action` and value `Auto-Stop`.
- Tag the second instance with a key `Action` and value `Auto-Start`.



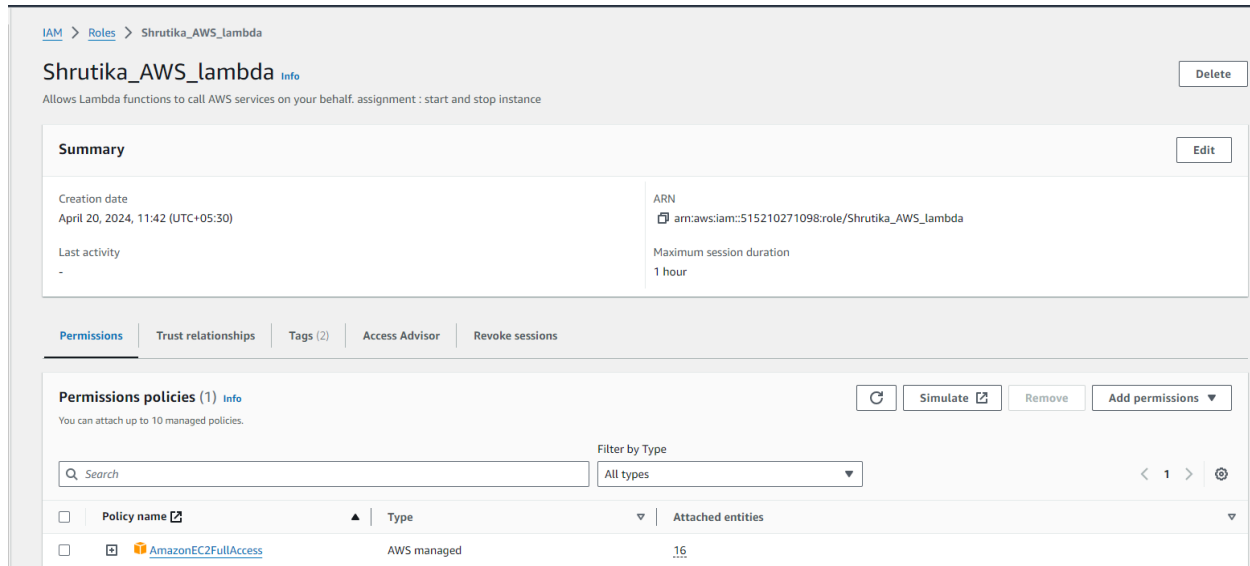
The screenshot shows the AWS Management Console 'Instances' page. At the top, there are buttons for 'Connect', 'Instance state', 'Actions', and 'Launch instances'. Below these is a search bar and filter buttons for 'Name = SP_Running' and 'Name = SP_Stopped'. The main table lists two instances:

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
<input type="checkbox"/>	SP_Stopped	i-01dd88e85476c152e	Stopped	t2.micro	-	View alarms +	us-east-2c	-
<input type="checkbox"/>	SP_Running	i-0060b45a56416da89	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2c	ec2-18-22...

At the bottom, there is a 'Select an instance' dropdown and a close button.

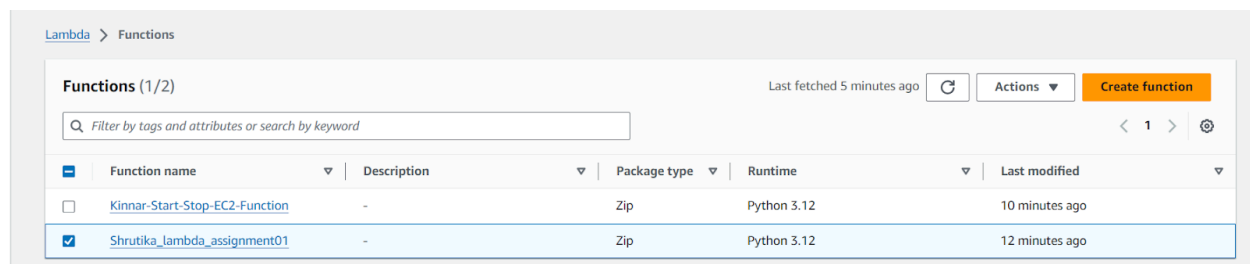
2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach the `AmazonEC2FullAccess` policy to this role. (Note: In a real-world scenario, you would want to limit permissions for better security.)



3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.
 - Go to the AWS Management Console and sign in with your AWS account credentials. Navigate to Lambda and Click on the "Create function" button.
 - Choose the option to author from scratch or use a blueprint. For your case, you'll be authoring from scratch.
 - Choose the runtime. Since you mentioned Python 3.x, select the Python runtime version you prefer.
 - Choose an existing role or create a new role with permissions for your Lambda function. Since you're working with EC2 instances, you'll need permissions to describe EC2 instances and perform actions on them. Make sure the role you choose or create has the necessary permissions.
 - If you need more granular control over permissions, you can create a custom IAM role with specific policies attached to it.



- Write the Boto3 Python script to:

1. Initialize a boto3 EC2 client.

2. Describe instances with `Auto-Stop` and `Auto-Start` tags.
3. Stop the `Auto-Stop` instances and start the `Auto-Start` instances.
4. Print instance IDs that were affected for logging purposes.

Lambda > Functions > Shrutika_lambda_assignment01

Shrutika_lambda_assignment01

Throttle Copy ARN Actions

Function overview Info

Export to Application Composer Download

Diagram Template

Shrutika_lambda_assignment01

Layers (0)

+ Add trigger + Add destination

Description

Last modified 13 minutes ago

Function ARN
arn:aws:lambda:us-east-2:515210271098:function:Shrutika_lambda_assignment01

Function URL Info

Code Test Monitor Configuration Aliases Versions

Code Test Monitor Configuration Aliases Versions

Code source Info

Upload from

File Edit Find View Go Tools Window Test Deploy

Go to Anything (Ctrl-P)

Environment

Shrutika_lambda_assignment01

lambda_function.py

```
1 import boto3
2
3 def lambda_handler(event, context):
4     # Initialize Boto3 EC2 client
5     ec2_client = boto3.client('ec2')
6
7     # Describe instances with Auto-Stop and Auto-Start tags
8     response = ec2_client.describe_instances(Filters=[
9         {
10             'Name': 'tag:Action',
11             'Values': ['Auto-Stop', 'Auto-Start']
12         }
13     ])
14
15     # Initialize lists to store instance IDs for logging purposes
16     stopped_instances = ['i-0060b45a56416da89']
17     started_instances = ['i-01dd88e85476c152e']
18
19     # Loop through instances and perform actions based on their tags
20     for reservation in response['Reservations']:
21         for instance in reservation['Instances']:
22             instance_id = instance['InstanceId']
23             for tag in instance['Tags']:
24                 if tag['Key'] == 'Action':
25                     action = tag['Value']
26                     if action == 'Auto-Stop' and instance['State']['Name'] == 'running':
27                         # Stop the Auto-Stop instances if they are currently running
28                         ec2_client.stop_instances(InstanceIds=[instance_id])
29                         stopped_instances.append(instance_id)
```

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Python Script :

```

Assignment01.py > lambda_handler
1  import boto3
2
3  def lambda_handler(event, context):
4      # Initialize Boto3 EC2 client
5      ec2_client = boto3.client('ec2')
6
7      # Describe instances with Auto-Stop and Auto-Start tags
8      response = ec2_client.describe_instances(Filters=[
9          {
10             'Name': 'tag:Action',
11             'Values': ['Auto-Stop', 'Auto-Start']
12         }
13     ])
14
15     # Initialize lists to store instance IDs for logging purposes
16     stopped_instances = ['i-0060b45a56416da89']
17     started_instances = ['i-01dd88e85476c152e']
18
19     # Loop through instances and perform actions based on their tags
20     for reservation in response['Reservations']:
21         for instance in reservation['Instances']:
22             instance_id = instance['InstanceId']
23             for tag in instance['Tags']:
24                 if tag['Key'] == 'Action':
25                     action = tag['Value']
26                     if action == 'Auto-Stop' and instance['State']['Name'] == 'running':
27                         # Stop the Auto-Stop instances if they are currently running
28                         ec2_client.stop_instances(InstanceIds=[instance_id])
29                         stopped_instances.append(instance_id)
30                     elif action == 'Auto-Start' and instance['State']['Name'] == 'stopped':
31                         # Start the Auto-Start instances if they are currently stopped
32                         ec2_client.start_instances(InstanceIds=[instance_id])
33                         started_instances.append(instance_id)
34
35     # Print instance IDs that were affected for logging purposes
36     print("Instances stopped:", stopped_instances)
37     print("Instances started:", started_instances)
38
39     # For testing locally
40     if __name__ == "__main__":
41         lambda_handler(None, None)

```

4. Manual Invocation:

- After saving your function, manually trigger it.
- Go to the EC2 dashboard and confirm that the instances' states have changed according to their tags.

lambda_function × Environment Var × Execution result ×	
▼ Execution results	
Test Event Name	testing
Response	null
Function Logs	
START RequestId: 62eccbd3-f3db-4a98-aba8-7bbc9c010ece Version: \$LATEST	
Instances stopped: ['i-0060b45a56416da89', 'i-0060b45a56416da89']	
Instances started: ['i-01dd88e85476c152e', 'i-01dd88e85476c152e']	
END RequestId: 62eccbd3-f3db-4a98-aba8-7bbc9c010ece	
REPORT RequestId: 62eccbd3-f3db-4a98-aba8-7bbc9c010ece Duration: 4026.89 ms Billed Duration: 4027 ms Memory Size: 128 MB Max Memory Used: 89 MB Init Duration: 271	
Request ID	
62eccbd3-f3db-4a98-aba8-7bbc9c010ece	

Instances (2) Info								
<input type="text" value="Find Instance by attribute or tag (case-sensitive)"/> <input type="button" value="All states"/>								
<input type="button" value="Name = SP_Running"/> <input type="button" value="Name = SP_Stopped"/> <input type="button" value="Clear filters"/>								
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
<input type="checkbox"/>	SP_Stopped	i-01dd88e85476c152e	Running	t2.micro	Initializing	View alarms +	us-east-2c	ec2-18-
<input type="checkbox"/>	SP_Running	i-0060b45a56416da89	Stopped	t2.micro	-	View alarms +	us-east-2c	-

Assignment 2 :

Automated S3 Bucket Cleanup Using AWS Lambda and Boto3.

To gain experience with AWS Lambda and Boto3 by creating a Lambda function that will automatically clean up old files in an S3 bucket.

1. S3 Setup:

- Navigate to the S3 dashboard and create a new bucket.

Amazon S3 > Buckets			
<div> <div>Account snapshot</div> <div>All AWS Regions</div> <div>View Storage Lens dashboard</div> </div> <div> <div>Last updated: Apr 19, 2024 by Storage Lens. Metrics are generated every 24 hours. Metrics don't include directory buckets. Learn more</div> </div>			
Total storage	Object count	Average object size	You can enable advanced metrics in the "default-account-dashboard" configuration.
5.1 GB	11.1 k	484.8 KB	
<div>General purpose buckets</div> <div>Directory buckets</div>			
<div>General purpose buckets (46)</div> <div>All AWS Regions</div> <div>Copy ARN</div> <div>Empty</div> <div>Delete</div> <div>Create bucket</div>			
<div>Buckets are containers for data stored in S3.</div> <div> <input type="text" value="shrutika"/> <div>1 match</div> </div>			
Name	AWS Region	IAM Access Analyzer	Creation date
bucket-shrutika	US East (Ohio) us-east-2	View analyzer for us-east-2	April 20, 2024, 13:04:23 (UTC+05:30)

- Upload multiple files to this bucket, ensuring that some files are older than 30 days (you may need to adjust your system's date temporarily for this or use old files).

Summary

Destination

s3://bucket-shrutika

Succeeded

✔ 3 files, 592.0 KB (100.00%)

Failed

🚫 0 files, 0 B (0%)

Files and folders

Configuration

Files and folders (3 Total, 592.0 KB)

🔍 Find by name

< 1 >

Name	Folder	Type	Size	Status	Error
WhatsApp Image 2024-04-14 ...	-	image/jpeg	174.8 KB	✔ Succeeded	-
WhatsApp Image 2023-04-30 ...	-	image/jpeg	223.5 KB	✔ Succeeded	-
WhatsApp Image 2023-04-30 ...	-	image/jpeg	193.6 KB	✔ Succeeded	-

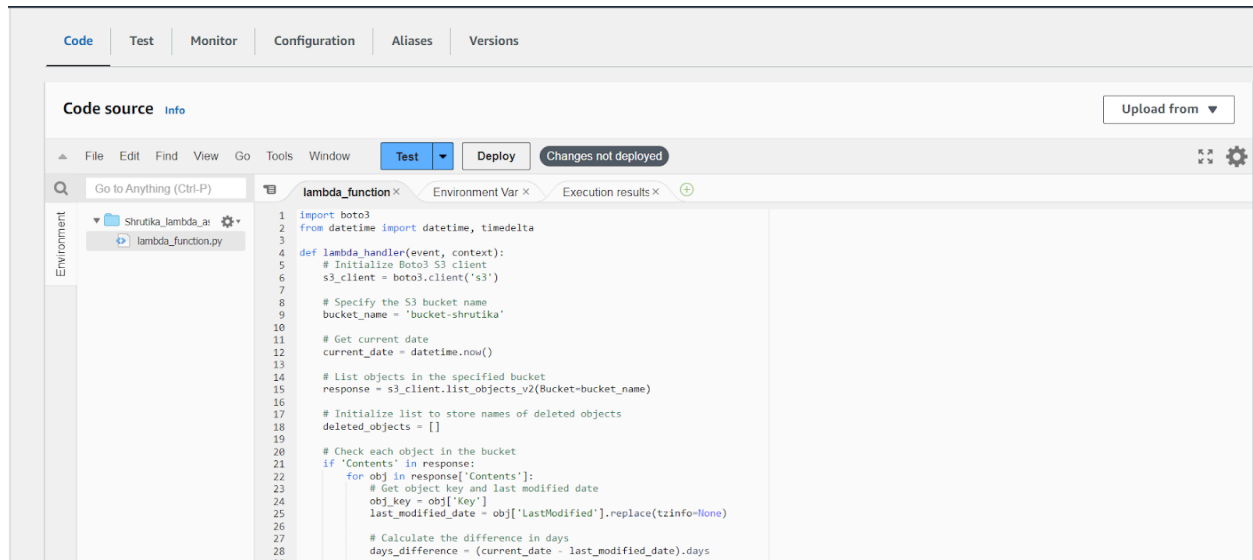
(Steps 2&3 will be same as of first assignment)

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach the `AmazonS3FullAccess` policy to this role. (Note: For enhanced security in real-world scenarios, use more restrictive permissions.)

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.
- Choose Python 3.x as the runtime.
- Assign the IAM role created in the previous step.
- Write the Boto3 Python script to:
 1. Initialize a boto3 S3 client.
 2. List objects in the specified bucket.
 3. Delete objects older than 30 days.
 4. Print the names of deleted objects for logging purposes.



Python Script :

```
1 import boto3
2 from datetime import datetime, timedelta
3
4 def lambda_handler(event, context):
5     # Initialize Boto3 S3 client
6     s3_client = boto3.client('s3')
7
8     # Specify the S3 bucket name
9     bucket_name = 'bucket-shrutika'
10
11     # Get current date
12     current_date = datetime.now()
13
14     # List objects in the specified bucket
15     response = s3_client.list_objects_v2(Bucket=bucket_name)
16
17     # Initialize list to store names of deleted objects
18     deleted_objects = []
19
20     # Check each object in the bucket
21     if 'Contents' in response:
22         for obj in response['Contents']:
23             # Get object key and last modified date
24             obj_key = obj['Key']
25             last_modified_date = obj['LastModified'].replace(tzinfo=None)
26
27             # Calculate the difference in days
28             days_difference = (current_date - last_modified_date).days
29
30             # Delete objects older than 30 days
31             if days_difference > 30:
32                 s3_client.delete_object(Bucket=bucket_name, Key=obj_key)
33                 deleted_objects.append(obj_key)
34
35     # Print the names of deleted objects for logging purposes
36     print("Deleted objects:", deleted_objects)
```

4. Manual Invocation:

- After saving your function, manually trigger it.
- Go to the S3 dashboard and confirm that only files newer than 30 days remain.

Assignment 3:

Automated RDS Snapshot Using AWS Lambda and Boto3.

To become familiar with automating RDS tasks using AWS Lambda and Boto3. You will create a Lambda function that takes a snapshot of an RDS instance.

What is RDS :

Amazon RDS (Relational Database Service) is a managed relational database service provided by Amazon Web Services (AWS). It enables you to easily set up, operate, and scale relational databases in the cloud without managing the infrastructure.

Key features of Amazon RDS include:

- **Managed Service:** Amazon RDS handles routine database tasks such as provisioning, patching, backup, recovery, and scaling, allowing you to focus on your application development instead of managing database infrastructure.
- **Multiple Database Engines:** Amazon RDS supports several popular database engines, including Amazon Aurora (a MySQL and PostgreSQL-compatible database engine developed by AWS), MySQL, PostgreSQL, MariaDB, Oracle, and Microsoft SQL Server. This enables you to choose the database engine that best fits your application requirements.
- **Automatic Backups and Point-in-Time Recovery:** Amazon RDS automatically backs up your database and retains backups for a specified retention period. You can also perform point-in-time recovery to restore your database to any specific point within the retention period.
- **High Availability and Fault Tolerance:** Amazon RDS provides high availability and fault tolerance for your databases by offering Multi-AZ (Multi-Availability Zone) deployments. In a Multi-AZ deployment, Amazon RDS automatically replicates your database across multiple Availability Zones within a region to provide redundancy and failover capabilities.
- **Scalability:** Amazon RDS allows you to easily scale your database instance vertically (by adjusting instance size) or horizontally (by adding read replicas). This enables you to handle increasing workloads and traffic demands without downtime.
- **Security:** Amazon RDS integrates with AWS Identity and Access Management (IAM) to control access to your databases. It also supports encryption at rest and in transit to enhance data security.

- **Monitoring and Metrics:** Amazon RDS provides built-in monitoring and metrics through Amazon CloudWatch, allowing you to monitor database performance, set alarms, and troubleshoot issues.

1. RDS Setup:

- Navigate to the RDS dashboard and create a new RDS instance (use the free tier, if available).
- Note the name of the instance.

The screenshot shows the Amazon RDS console for a database instance named 'database-1'. The breadcrumb navigation at the top is 'RDS > Databases > database-1'. The instance details are as follows:

DB identifier	Status	Role	Engine	Region & AZ	Size	Recommendations	CPU	Current act
database-1	Available	Regional cluster	Aurora PostgreSQL	us-east-2	2 instances	-	-	-
database-1-instance-1	Available	Writer instance	Aurora PostgreSQL	us-east-2b	db.r6g.2xlarge		2.00%	0.0
database-1-instance-1-us-east-2c	Available	Reader instance	Aurora PostgreSQL	us-east-2c	db.r6g.2xlarge		2.24%	0.0

Below the instance details, there are tabs for 'Connectivity & security', 'Monitoring', 'Logs & events', 'Configuration', 'Maintenance & backups', 'Tags', and 'Recommendations'. The 'Endpoints (2)' section is expanded, showing two endpoints:

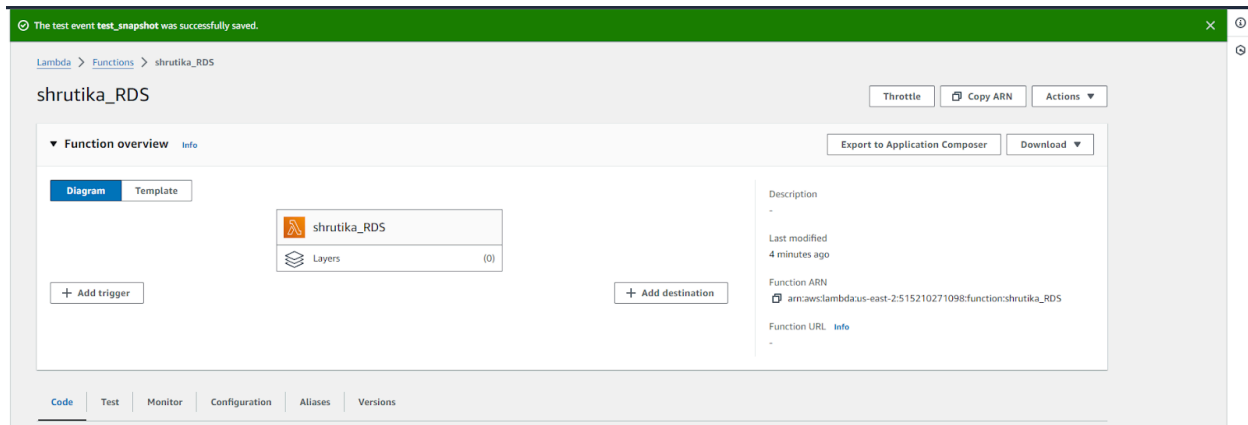
Endpoint name	Status	Type	Port
database-1.cluster-czyfftv2dgcus-east-2.rds.amazonaws.com	Available	Writer	5432
database-1.cluster-ro-czyfftv2dgcus-east-2.rds.amazonaws.com	Available	Reader	5432

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach the `AmazonRDSFullAccess` policy to this role. (Note: Always practice the principle of least privilege in real-world scenarios.)

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.



- Choose Python 3.x as the runtime.
- Assign the IAM role created in the previous step.

- Write the Boto3 Python script to:

1. Initialize a boto3 RDS client.
2. Take a snapshot of the specified RDS instance.
3. Print the snapshot ID for logging purposes.

```

1  import boto3
2
3  def take_rds_snapshot(instance_identifier):
4      # Initialize Boto3 RDS client
5      rds_client = boto3.client('rds')
6
7      # Take a snapshot of the specified RDS instance
8      response = rds_client.create_db_snapshot(
9          DBSnapshotIdentifier='snapshot-' + instance_identifier.replace('.', '-'), # Example
10         DBInstanceIdentifier=instance_identifier
11     )
12
13     # Print the snapshot ID for logging purposes
14     print("Snapshot ID:", response['DBSnapshot']['DBSnapshotIdentifier'])
15
16 # Example usage: Specify your RDS instance identifier
17 instance_identifier = 'database-1-instance-1.czyfftv2dgcu.us-east-2.rds.amazonaws.com'
18 take_rds_snapshot(instance_identifier)

```

4. Event Source (Bonus):

- Attach an event source, like Amazon CloudWatch Events, to trigger the Lambda function every day (or as per your preferred frequency).

[Amazon EventBridge](#) > [Schedules](#) > RDS_snapshot

RDS_snapshot

Disable

Edit

Delete

Schedule detail

Schedule name RDS_snapshot	Status ✔ Enabled	Schedule start time -	Flexible time window 5 minutes
Description -	Schedule ARN arn:aws:scheduler:us-east-2:515210271098:schedule/default/RDS_snapshot	Schedule end time -	Created date Apr 20, 2024, 14:24:24 (UTC+05:30)
Schedule group name default	Execution time zone Asia/Calcutta	Action after completion NONE	Last modified date Apr 20, 2024, 14:24:24 (UTC+05:30)

Schedule

Target

Retry policy

Dead-letter queue

Encryption

Schedule

Cron expression

Info

0

0

*

*

?

*

Minutes

Hours

Day of month

Month

Day of week

Year

Copy cron expression

Next 10 trigger dates

Date and time are displayed in the selected time zone for which this schedule is set in UTC format, e.g. "Wed, Nov 9, 2022 09:00 (UTC - 08:00)"

Sun, 21 Apr 2024 00:00:00 (UTC+05:30)

Mon, 22 Apr 2024 00:00:00 (UTC+05:30)

Tue, 23 Apr 2024 00:00:00 (UTC+05:30)

Wed, 24 Apr 2024 00:00:00 (UTC+05:30)

Thu, 25 Apr 2024 00:00:00 (UTC+05:30)

Fri, 26 Apr 2024 00:00:00 (UTC+05:30)

5. Manual Invocation:

- After saving your function, manually trigger it (or wait for the scheduled trigger).
- Go to the RDS dashboard and confirm that a snapshot has been taken.

Assignment 4:

Monitor Unencrypted S3 Buckets Using AWS Lambda and Boto3.

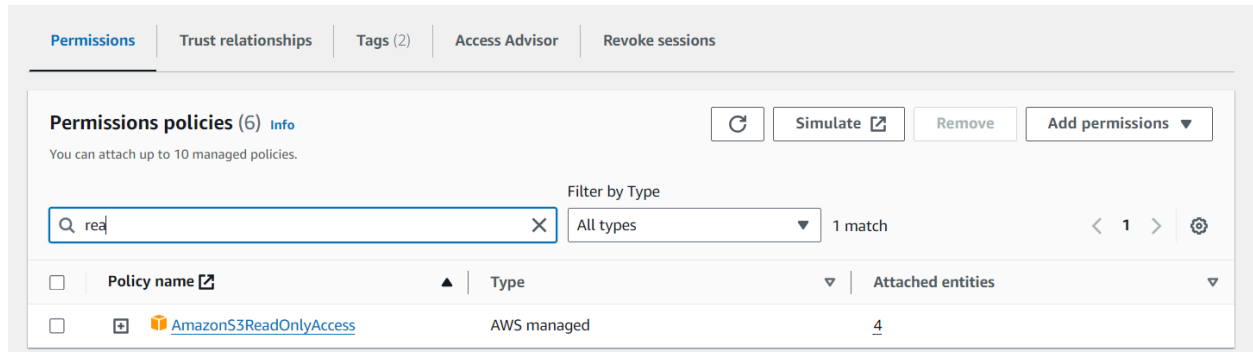
To enhance your AWS security posture by setting up a Lambda function that detects any S3 bucket without server-side encryption.

1. S3 Setup:

- Navigate to the S3 dashboard and create a few buckets. Ensure that a couple of them don't have server-side encryption enabled.

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach the `AmazonS3ReadOnlyAccess` policy to this role.



3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.
 - Choose Python 3.x as the runtime.
 - Assign the IAM role created in the previous step.
- Write the Boto3 Python script to:
1. Initialize a boto3 S3 client.
 2. List all S3 buckets.
 3. Detects buckets without server-side encryption.
 4. Print the names of unencrypted buckets for logging purposes.

```

1  import boto3
2
3  def detect_unencrypted_buckets():
4      # Initialize Boto3 S3 client
5      s3_client = boto3.client('s3')
6
7      try:
8          # List all S3 buckets
9          response = s3_client.list_buckets()
10
11         # Detect buckets without server-side encryption
12         unencrypted_buckets = []
13         for bucket in response['Buckets']:
14             bucket_name = bucket['Name']
15             bucket_encryption = s3_client.get_bucket_encryption(Bucket=bucket_name)
16             if 'ServerSideEncryptionConfiguration' not in bucket_encryption:
17                 unencrypted_buckets.append(bucket_name)
18
19         # Print the names of unencrypted buckets for logging purposes
20         if unencrypted_buckets:
21             print("Unencrypted buckets:")
22             for bucket_name in unencrypted_buckets:
23                 print(bucket_name)
24         else:
25             print("No unencrypted buckets found.")
26
27     except Exception as e:
28         print("Error:", str(e))
29
30 # Invoke the function to detect unencrypted buckets
31 detect_unencrypted_buckets()

```

4. Manual Invocation:

- After saving your function, manually trigger it.
- Review the Lambda logs to identify the buckets without server-side encryption.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\Shrutika\Desktop\boto3> & C:/Users/Shrutika/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Shrutika/Desktop/boto3/test_boto.py
No unencrypted buckets found.
PS C:\Users\Shrutika\Desktop\boto3>

```

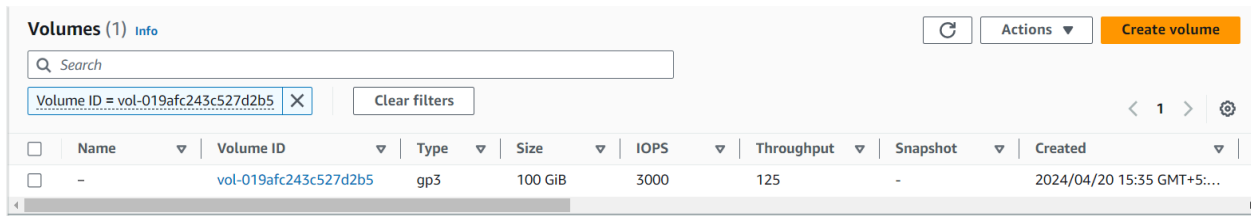
Assignment 5:

Automatic EBS Snapshot and Cleanup Using AWS Lambda and Boto3

To automate the backup process for your EBS volumes and ensure that backups older than a specified retention period are cleaned up to save costs.

1. EBS Setup:

- Navigate to the EC2 dashboard and identify or create an EBS volume you wish to back up.
- Note down the volume ID.



Volumes (1) Info

Search

Volume ID = vol-019afc243c527d2b5 X Clear filters

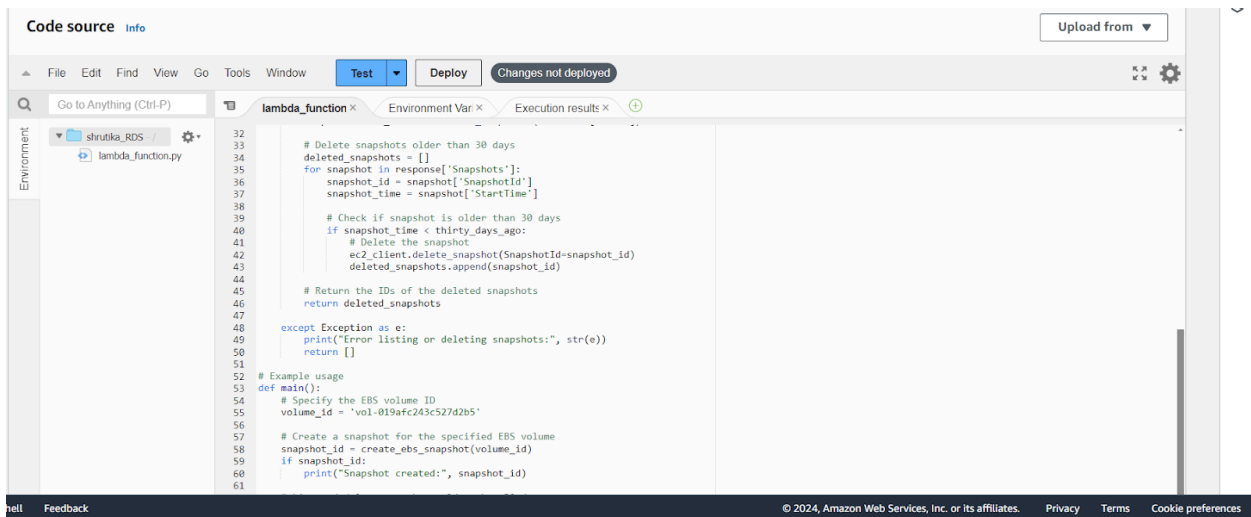
	Name	Volume ID	Type	Size	IOPS	Throughput	Snapshot	Created
<input type="checkbox"/>	-	vol-019afc243c527d2b5	gp3	100 GiB	3000	125	-	2024/04/20 15:35 GMT+5...

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach policies that allow Lambda to create EBS snapshots and delete them ('AmazonEC2FullAccess' for simplicity, but be more restrictive in real-world scenarios).

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.
- Choose Python 3.x as the runtime.
- Assign the IAM role created in the previous step.
- Write the Boto3 Python script to:
 1. Initialize a boto3 EC2 client.
 2. Create a snapshot for the specified EBS volume.
 3. List snapshots and delete those older than 30 days.
 4. Print the IDs of the created and deleted snapshots for logging purposes.



Code source Info

Upload from

File Edit Find View Go Tools Window Test Deploy Changes not deployed

Go to Anything (Ctrl-P)

Environment

shrutika_RDS / lambda_function.py

```

32
33
34 # Delete snapshots older than 30 days
35 deleted_snapshots = []
36 for snapshot in response['Snapshots']:
37     snapshot_id = snapshot['SnapshotId']
38     snapshot_time = snapshot['StartTime']
39
40     # Check if snapshot is older than 30 days
41     if snapshot_time < thirty_days_ago:
42         # Delete the snapshot
43         ec2_client.delete_snapshot(SnapshotId=snapshot_id)
44         deleted_snapshots.append(snapshot_id)
45
46     # Return the IDs of the deleted snapshots
47     return deleted_snapshots
48
49 except Exception as e:
50     print("Error listing or deleting snapshots:", str(e))
51     return []
52
53 # Example usage
54 def main():
55     # Specify the EBS volume ID
56     volume_id = 'vol-019afc243c527d2b5'
57
58     # Create a snapshot for the specified EBS volume
59     snapshot_id = create_ebs_snapshot(volume_id)
60     if snapshot_id:
61         print("Snapshot created:", snapshot_id)

```

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Python Script :

```
1  import boto3
2  from datetime import datetime, timedelta
3
4  def create_ebs_snapshot(volume_id):
5      # Initialize Boto3 EC2 client
6      ec2_client = boto3.client('ec2')
7
8      try:
9          # Create a snapshot for the specified EBS volume
10         response = ec2_client.create_snapshot(
11             VolumeId=volume_id,
12             Description='Snapshot created by backup script'
13         )
14
15         # Return the snapshot ID
16         return response['SnapshotId']
17
18     except Exception as e:
19         print("Error creating snapshot:", str(e))
20         return None
21
22 def list_and_delete_old_snapshots():
23     # Initialize Boto3 EC2 client
24     ec2_client = boto3.client('ec2')
25
26     try:
27         # Calculate the date 30 days ago
28         thirty_days_ago = datetime.now() - timedelta(days=30)
29
30         # List all snapshots
31         response = ec2_client.describe_snapshots(OwnerIds=['self'])
32
33         # Delete snapshots older than 30 days
34         deleted_snapshots = []
35         for snapshot in response['Snapshots']:
36             snapshot_id = snapshot['SnapshotId']
37             snapshot_time = snapshot['StartTime']
38
39             # Check if snapshot is older than 30 days
40             if snapshot_time < thirty_days_ago:
41                 # Delete the snapshot
42                 ec2_client.delete_snapshot(SnapshotId=snapshot_id)
43                 deleted_snapshots.append(snapshot_id)
44
45         # Return the IDs of the deleted snapshots
46         return deleted_snapshots
47
48     except Exception as e:
49         print("Error listing or deleting snapshots:", str(e))
50         return []
51
52 # Example usage
53 def main():
54     # Specify the EBS volume ID
55     volume_id = 'vol-019afc243c527d2b5'
56
57     # Create a snapshot for the specified EBS volume
58     snapshot_id = create_ebs_snapshot(volume_id)
59     if snapshot_id:
60         print("Snapshot created:", snapshot_id)
61
62     # List and delete snapshots older than 30 days
63     deleted_snapshots = list_and_delete_old_snapshots()
64     if deleted_snapshots:
65         print("Snapshots deleted:", deleted_snapshots)
66
67 if __name__ == "__main__":
68     main()
```

4. Event Source (Bonus):

- Attach an event source, like Amazon CloudWatch Events, to trigger the Lambda function at your desired backup frequency (e.g., every week).

5. Manual Invocation:

- After saving your function, either manually trigger it or wait for the scheduled event.
- Go to the EC2 dashboard and confirm that the snapshot is created and old snapshots are deleted.

```
PS C:\Users\Shrutika\Desktop\boto3> & c:/Users/Shrutika/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Shrutika/Desktop/boto3/test_boto.py
Snapshot created: snap-0e2a5d5462d0010bb
Error listing or deleting snapshots: can't compare offset-naive and offset-aware datetimes
PS C:\Users\Shrutika\Desktop\boto3> & c:/Users/Shrutika/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Shrutika/Desktop/boto3/test_boto.py
```

Assignment 6:

Auto-Tagging EC2 Instances on Launch Using AWS Lambda and Boto3.

Learn to automate the tagging of EC2 instances as soon as they are launched, ensuring better resource tracking and management.

1. EC2 Setup:

- Ensure you have the capability to launch EC2 instances.

2. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach the `AmazonEC2FullAccess` policy to this role.

3. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.
- Choose Python 3.x as the runtime.
- Assign the IAM role created in the previous step.
- Write the Boto3 Python script to:
 1. Initialize a boto3 EC2 client.
 2. Retrieve the instance ID from the event.
 3. Tag the new instance with the current date and another tag of your choice.
 4. Print a confirmation message for logging purposes.


```

1  import boto3
2  import datetime
3
4  def tag_instance(event):
5      # Initialize a boto3 EC2 client
6      ec2_client = boto3.client('ec2')
7
8      # Retrieve the instance ID from the event
9      instance_id = event['instance_id']
10
11     # Tag the new instance with the current date and another tag
12     current_date = datetime.datetime.now().strftime('%Y-%m-%d')
13     tag_key = 'DateCreated'
14     tag_value = current_date
15     another_tag_key = 'Test_Tag'
16     another_tag_value = 'Test_value'
17     tags = [
18         {'key': tag_key, 'Value': tag_value},
19         {'key': another_tag_key, 'Value': another_tag_value}
20     ]
21     ec2_client.create_tags(Resources=[instance_id], Tags=tags)
22
23     # Print a confirmation message for logging purposes
24     print(f"Instance {instance_id} tagged with {tag_key}: {tag_value} and {another_tag_key}: {another_tag_value}")
25
26     # Example event containing the instance ID for manual testing
27     event = {'instance_id': 'i-074cbdef4dbe31b1a'}
28
29     # Call the function with the event
30     tag_instance(event)

```

4. CloudWatch Events:

- Set up a CloudWatch Event Rule to trigger the EC2 instance launch event.
- Attach the Lambda function as the target.

5. Testing:

- Launch a new EC2 instance.
- After a short delay, confirm that the instance is automatically tagged as specified.

```
1 import boto3
2 import datetime
3
4 def tag_instance(event):
5     # Initialize a boto3 EC2 client
6     ec2_client = boto3.client('ec2')
7
8     # Retrieve the instance ID from the event
9     instance_id = event['instance_id']
10
11     # Tag the new instance with the current date and another tag
12     current_date = datetime.datetime.now().strftime('%Y-%m-%d')
13     tag_key = 'DateCreated'
14     tag_value = current_date
15     another_tag_key = 'Assignment'
16     another_tag_value = 'Test_EC2'
17     tags = [
18         {'Key': tag_key, 'Value': tag_value},
19         {'Key': another_tag_key, 'Value': another_tag_value}
20     ]
21     ec2_client.create_tags(Resources=[instance_id], Tags=tags)
22
23     # Print a confirmation message for logging purposes
24     print(f"Instance {instance_id} tagged with {tag_key}: {tag_value} and {another_tag_key}: {another_tag_value}")
25
26 # Example event containing the instance ID for manual testing
27 event = {'instance_id': 'i-0eb63e35e0e21c24d'}
28
29 # Call the function with the event
30 tag_instance(event)
```

[EC2](#) > [Instances](#) > [i-074cbdef4dbe31b1a](#) > **Manage tags**

Manage tags Info

A tag is a custom label that you assign to an AWS resource. You can use tags to help organize and identify your instances.

Key	Value - optional	
<input type="text" value="Test_Tag"/>	<input type="text" value="Test_value"/>	<button>Remove</button>
<input type="text" value="Name"/>	<input type="text" value="Shrutika_EC2"/>	<button>Remove</button>
<input type="text" value="DateCreated"/>	<input type="text" value="2024-04-21"/>	<button>Remove</button>

Add new tag

You can add up to 47 more tags.

CancelSave

Manual Testing :

```
PS C:\Users\Shrutika\Desktop\boto3> python .\test_boto.py
Instance i-074cbdef4dbe31b1a tagged with DateCreated: 2024-04-21 and Test_Tag: Test_value
PS C:\Users\Shrutika\Desktop\boto3> █
```

Assignment 9:

Analyze Sentiment of User Reviews Using AWS Lambda, Boto3, and Amazon Comprehend. Automatically analyze and categorize the sentiment of user reviews using Amazon Comprehend.

Task: Set up a Lambda function to receive user reviews, analyze their sentiment using Amazon Comprehend, and log the results.

Instructions:

1. Lambda IAM Role:

- In the IAM dashboard, create a new role for Lambda.
- Attach policies that allow Lambda to use Amazon Comprehend.

2. Lambda Function:

- Navigate to the Lambda dashboard and create a new function.
- Choose Python 3.x as the runtime.
- Assign the IAM role created previously.
- Write the Boto3 Python script to:
 1. Extract the user review from an event.
 2. Use Amazon Comprehend to analyze the sentiment of the review.
 3. Log the sentiment result.

```

1  import boto3
2
3  # Initialize Boto3 clients
4  comprehend = boto3.client('comprehend')
5  cloudwatch = boto3.client('cloudwatch')
6
7  def extract_review(event):
8      # Assuming event contains the review text
9      review = event['review']
10     return review
11
12  def analyze_sentiment(review):
13      # Use Amazon Comprehend to analyze sentiment
14      response = comprehend.detect_sentiment(Text=review, LanguageCode='en')
15      sentiment = response['Sentiment']
16      return sentiment
17
18  def log_sentiment(sentiment):
19      # Log sentiment result to CloudWatch
20      response = cloudwatch.put_metric_data(
21          Namespace='SentimentAnalysis',
22          MetricData=[
23              {
24                  'MetricName': 'Sentiment',
25                  'Value': 1, # You can assign a numeric value to each sentiment if needed
26                  'Unit': 'Count',
27                  'Dimensions': [
28                      {
29                          'Name': 'SentimentType',
30                          'Value': sentiment
31                      }
32                  ]
33              }
34          ]
35      )
36
37  def main(event):
38      review = extract_review(event)
39      sentiment = analyze_sentiment(review)
40      log_sentiment(sentiment)
41
42  # Example usage
43  if __name__ == "__main__":
44      event = {'review': 'This product is amazing!'}
45      main(event)

```

3. Testing:

- Manually trigger the Lambda function with sample reviews.

Event name

test

↻

Delete

Event JSON

Format JSON

```

1  {
2    "review": "This product is good"
3  }

```

- Confirm the sentiment analysis results in the Lambda logs.

Assignment 11:

Notify When ELB 5xx Errors Spike Using AWS Lambda, Boto3, and SNS

To automatically receive notifications when your Elastic Load Balancer (ELB) encounters an unusually high number of 5xx errors.

Set up a Lambda function that checks the ELB's 5xx error metrics and sends an SNS notification if errors exceed a certain threshold.

Instructions:

1. SNS Setup:

- In the SNS dashboard, create a new topic.
- Subscribe your email or phone to this topic.

2. Lambda IAM Role:

- Create a new role for Lambda with permissions to read CloudWatch metrics and send SNS notifications.

3. Lambda Function:

- Create a function and assign the above IAM role.
- Use Boto3 to:
 1. Fetch the 5xx error count metric from CloudWatch for your ELB.
 2. If the count exceeds a threshold (e.g., 10 in the last 5 minutes), trigger an SNS notification.

```

1  import boto3
2  import datetime
3
4  def fetch_cloudwatch_metric(elb_name, duration_minutes):
5      # Create CloudWatch client
6      cloudwatch = boto3.client('cloudwatch')
7
8      # Define time range for the metric query
9      end_time = datetime.datetime.now()
10     start_time = end_time - datetime.timedelta(minutes=duration_minutes)
11
12     # Fetch the metric data
13     response = cloudwatch.get_metric_statistics(
14         Namespace='AWS/ELB',
15         MetricName='HTTPCode_Backend_5XX',
16         Dimensions=[
17             {
18                 'Name': 'LoadBalancerName',
19                 'Value': elb_name
20             },
21         ],
22         StartTime=start_time,
23         EndTime=end_time,
24         Period=60, # 1-minute granularity
25         Statistics=['Sum']
26     )
27
28     # Calculate the total count of 5xx errors
29     error_count = 0
30     for datapoint in response['Datapoints']:
31         error_count += datapoint['Sum']
32
33     return error_count
34
35 def trigger_sns_notification(topic_arn, count_threshold, duration_minutes):
36     # Fetch the 5xx error count metric for your ELB
37     elb_name = 'LBAssignmentSP'
38     error_count = fetch_cloudwatch_metric(elb_name, duration_minutes)
39
40     # If the count exceeds the threshold, trigger an SNS notification
41     if error_count > count_threshold:
42         sns = boto3.client('sns')
43         message = f"5xx error count for ELB '{elb_name}' exceeded threshold ({count_threshold}) in the last {duration_minutes} minutes. Count: {error_count}"
44         sns.publish(
45             TopicArn=topic_arn,
46             Message=message,
47             Subject=f"5xx Error Count Exceeded Threshold for ELB '{elb_name}'"
48         )
49
50 def main():
51     # Specify your SNS topic ARN and threshold values
52     topic_arn = 'arn:aws:sns:us-east-2:515210271098:EC2_State_Change_Notifications'
53     count_threshold = 10
54     duration_minutes = 5
55
56     # Trigger SNS notification if error count exceeds threshold
57     trigger_sns_notification(topic_arn, count_threshold, duration_minutes)
58
59 if __name__ == "__main__":
60     main()

```

4. CloudWatch Events:

- Schedule your Lambda function to run every 5 minutes.

5. Testing:

- Simulate or wait for a spike in 5xx errors.

Manual on local :

```
PS C:\Users\Shrutika\Desktop\boto3> python .\revies.py
Error count is within threshold.
PS C:\Users\Shrutika\Desktop\boto3> █
```

- Confirm receipt of the SNS notification.

Assignment 14:

Audit S3 Bucket Permissions and Notify for Public Buckets

Automatically audit S3 bucket permissions and send notifications if any buckets have public read or write permissions.

Set up a Lambda function to regularly audit S3 bucket permissions and send SNS notifications for any buckets that are publicly accessible.

Instructions:

1. SNS Setup:

- Create a new SNS topic.
- Subscribe to the topic with your email.

2. Lambda IAM Role:

- Create a role with permissions to list S3 bucket permissions and send SNS notifications.

3. Lambda Function:

- Create a function with the above IAM role.
- Use Boto3 to:
 1. List all S3 buckets.
 2. For each bucket, check its permission settings.
 3. If a bucket has public read or write permissions, send a notification with its name via SNS.

```

1  import boto3
2  from botocore.exceptions import ClientError
3
4  def list_s3_buckets():
5      # Mocked list of bucket names for testing
6      return ['bucket1', 'bucket2', 'bucket3']
7
8  def check_bucket_permissions(bucket_name):
9      # Simulate checking permissions for each bucket
10     # For testing purposes, assume 'bucket1' has public write permission
11     if bucket_name == 'bucket1':
12         return True
13     return False
14
15  def send_notification(bucket_name):
16      # Print notification message for testing
17      print(f"Bucket '{bucket_name}' has public read or write permissions.")
18
19  def main():
20      # List all S3 buckets
21      bucket_names = list_s3_buckets()
22
23      # Check permissions for each bucket
24      for bucket_name in bucket_names:
25          if check_bucket_permissions(bucket_name):
26              # If public access is detected, print notification
27              send_notification(bucket_name)
28
29  if __name__ == "__main__":
30      main()
31

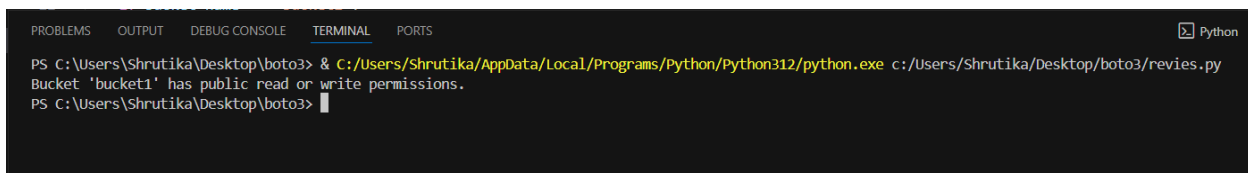
```

4. CloudWatch Events:

- Schedule your Lambda function to run daily.

5. Testing:

- Make one or two of your S3 buckets public.
- Run the Lambda function and ensure you receive appropriate SNS notifications.



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python
PS C:\Users\Shrutika\Desktop\boto3> & C:/Users/Shrutika/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Shrutika/Desktop/boto3/revies.py
Bucket 'bucket1' has public read or write permissions.
PS C:\Users\Shrutika\Desktop\boto3>

```

Assignment 15:

Monitor EC2 Instance State Changes Using AWS Lambda, Boto3, and SNS

Automatically monitor changes in EC2 instance states and send notifications whenever an instance is started or stopped.

Set up a Lambda function that listens to EC2 state change events and sends SNS notifications detailing the state changes.

Instructions:

1. SNS Setup:

- Navigate to the SNS dashboard and create a new topic.
- Subscribe to this topic with your email.

The image shows two screenshots from the Amazon SNS console. The top screenshot is the 'Create subscription' page, and the bottom screenshot is the 'Topic details' page for 'EC2_State_Change_Notifications'.

Create subscription

Amazon SNS > Subscriptions > Create subscription

Details

Topic ARN
arn:aws:sns:us-east-2:515210271098:EC2_State_Change_Notifications

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
panchalshrutika00@gmail.com

After your subscription is created, you must confirm it. [Info](#)

EC2_State_Change_Notifications

Amazon SNS > Topics > EC2_State_Change_Notifications

Edit Delete Publish message

Details

Name EC2_State_Change_Notifications	Display name SP_EC2_State_Change_Notifications
ARN arn:aws:sns:us-east-2:515210271098:EC2_State_Change_Notifications	Topic owner 515210271098
Type Standard	

2. Lambda IAM Role:

- Create a role with permissions to read EC2 instance states and send SNS notifications.

3. Lambda Function:

- Create a function and assign the above IAM role.
- Use Boto3 to:
 1. Extract details from the event regarding the EC2 state change.

2. Send an SNS notification with details about which EC2 instance changed state and the new state (e.g., started, stopped).

```
1  import boto3
2
3  def extract_ec2_state_change_details(event):
4      # Extract relevant details from the event regarding the EC2 state change
5      instance_id = event['detail']['instance-id']
6      state = event['detail']['state']
7
8      return instance_id, state
9
10 def send_sns_notification(topic_arn, instance_id, state):
11     # Create SNS client
12     sns = boto3.client('sns')
13
14     # Compose message for the SNS notification
15     message = f"The state of EC2 instance {instance_id} has changed to {state}."
16
17     # Send SNS notification
18     sns.publish(
19         TopicArn=topic_arn,
20         Message=message,
21         Subject=f"EC2 Instance State Change: {instance_id}"
22     )
23
24 def main(event):
25     # Specify your SNS topic ARN
26     topic_arn = 'arn:aws:sns:us-east-2:515210271098:EC2_State_Change_Notifications'
27
28     # Extract details from the event regarding the EC2 state change
29     instance_id, state = extract_ec2_state_change_details(event)
30
31     # Send an SNS notification with details about the EC2 instance state change
32     send_sns_notification(topic_arn, instance_id, state)
```

4. EC2 Event Bridge (formerly CloudWatch Events):

- Set up an Event Bridge rule to trigger your Lambda function whenever an EC2 instance state changes.

5. Testing:

- Start or stop one of your EC2 instances.

EC2 > Instances > i-0ccafb92ade323d66

Instance summary for i-0ccafb92ade323d66 (Test_EC2) [Info](#)

Updated 37 minutes ago

[Refresh](#) [Connect](#) [Instance state ▼](#) [Actions ▼](#)

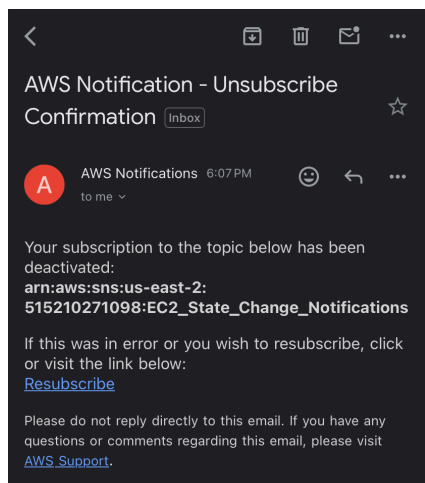
Instance ID i-0ccafb92ade323d66 (Test_EC2)	Public IPv4 address -	Private IPv4 addresses 172.31.39.47
IPv6 address -	Instance state Stopped	Public IPv4 DNS -
Hostname type IP name: ip-172-31-39-47.us-east-2.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-39-47.us-east-2.compute.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. Learn more
Auto-assigned IP address -	VPC ID vpc-7faf5614	Auto Scaling Group name -
IAM Role -	Subnet ID subnet-b00483fc	
IMDSv2 Required		

- Confirm you receive an SNS notification about the state change.

Subscriptions (2) [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

Search

	ID	Endpoint	Status	Protocol
<input type="radio"/>	8887565d-c587-4332-b26c-59e...	shrutikapanchal2000@gmail.com	Confirmed	EMAIL
<input checked="" type="radio"/>	Deleted	panchalshrutika00@gmail.com	Confirmed	EMAIL



Assignment 20: Load Balancer Health Checker

Design a Lambda function that checks the health of registered instances behind an Elastic Load Balancer (ELB) and notifies via SNS if any instances are unhealthy.

1. Create a Lambda function.

2. With Boto3, configure the function to:

- Check the health of registered instances behind a given ELB.
- If any instances are found to be unhealthy, publish a detailed message to an SNS topic.

```
1  import boto3
2
3  def check_instance_health(elb_name):
4      # Create ELB client
5      elb = boto3.client('elb')
6
7      # Describe instance health for the ELB
8      response = elb.describe_instance_health(
9          LoadBalancerName=elb_name
10     )
11
12     return response['InstanceStates']
13
14 def publish_sns_message(topic_arn, message):
15     # Create SNS client
16     sns = boto3.client('sns')
17
18     # Publish message to SNS topic
19     sns.publish(
20         TopicArn=topic_arn,
21         Message=message,
22         Subject="Unhealthy Instances Detected"
23     )
24
25 def main(elb_name, sns_topic_arn):
26     # Check the health of registered instances behind the ELB
27     instance_states = check_instance_health(elb_name)
28
29     # Check if any instances are unhealthy
30     unhealthy_instances = [instance for instance in instance_states if instance['State'] != 'InService']
31
32     # If unhealthy instances are found, publish a detailed message to the SNS topic
33     if unhealthy_instances:
34         message = f"Unhealthy instances detected behind ELB '{elb_name}':\n"
35         for instance in unhealthy_instances:
36             message += f"Instance ID: {instance['InstanceId']}, State: {instance['State']}\n"
37         publish_sns_message(sns_topic_arn, message)
38         print("Detailed message published to SNS topic.")
39     else:
40         print("All instances are healthy.")
41
42 if __name__ == "__main__":
43     # Specify your ELB name and SNS topic ARN
44     elb_name = 'Nikitasloadbalancer'
45     sns_topic_arn = 'arn:aws:sns:us-east-2:515210271098:EC2_State_Change_Notifications'
46
47     main(elb_name, sns_topic_arn)
```

3. Set up a CloudWatch event to trigger this Lambda function every 10 minutes.