Amazon EC2 AMI Backup Using Lambda Functions

Objective:

The main objective of this project is to Manage the EC2 AMI backup for EC2 instances using lambda functions in Python. Also deleting the EC2 AMIs any older than 30 days or more. Using python to automate this process. This project uses AWS EC2, AWS SNS, AWS CloudWatch and the programming language that will be used in this project is Python.

Functionality

For this Project, I am going to use the AWS and Python to manage and delete any older

EC2 AMIs in the AWS Environment. I will use lambda Function for backup and to delete the Backup based on the defined period. Lambda functions are scheduled to trigger at some time interval using CloudWatch events. Lambda Script for AMI will check for the current date on Every AMI taken and if it finds AMI older than 30 days, then it will delete those AMI’s.

This Process can take the backup of multiple EC2 Instances all together and will keep on deleting the Older AMI which no longer needs to be retained. Using SNS, the notifications are being sent to the registered email.

Implementation

1. Lambda Function

First a Lambda function was created. It works by checking all available instances in all regions. It then creates back up of the individual EBS volumes and stores the images in S3 and tags them with key ***backup*** and value ***true.***The reason for tagging is to ensure that only backups will be deleted during the deletion step below and not any other file(s). The backup files (AMIs) are named by combing the instance ID and the current timestamp.

Next, the function checks for the age of each image and deletes any image with tag ***backup:true*** and older than 30 days.

1. Roles and Policies

For the above lambda function to perform the required tasks, it must have sufficient permission to do so. I created an IAM role and granted it to the lambda function created above. I then created a policy with the required permissions and attached it to the IAM role previously created. Some of the permissions in the policy are list instances, describe instance, create image and delete snapshot. This is provided in the appendix at the end of the report.

1. Lambda Function Trigger.

For the above function to execute, it must be invoked. This is achieved by CloudWatch trigger which we scheduled to run on daily basis. Once triggered, the function performs as outlined in step 1 above.

1. Email Notification

I also created a CloudWatch rule to monitor creation and deletion of snapshot activities and then created a topic and subscription.

once the subscription created, we added the required email(s) for the users who required to be notified of creation or deletion of snapshots. We set the target of the CloudWatch rule above as the SNS topic created.

Whenever an event of snapshot creation or deletion occurs, the CloudWatch rule routes it to the target (SNS topic above). All the subscribers of the topic get notified.

Appendix

1. Lambda Function

import boto3

from datetime import datetime, timedelta

ec2 = boto3.client('ec2')

def lambda\_handler(event, context):

backup\_time = datetime.utcnow() + timedelta(minutes=5)

response = ec2.describe\_instances()

for reservation in response['Reservations']:

for instance in reservation['Instances']:

instance\_id = instance['InstanceId']

ami\_name = f"{instance\_id}-{backup\_time.strftime('%Y-%m-%d-%H-%M-%S')}"

print(ami\_name)

response = ec2.create\_image(InstanceId=instance\_id, Name=ami\_name+"test", NoReboot=True)

image\_id = response['ImageId']

ec2.create\_tags(Resources=[image\_id], Tags=[{'Key': 'backup', 'Value': 'true'}])

response = ec2.describe\_images(Filters=[{'Name': 'tag:backup', 'Values': ['true']}])

for image in response['Images']:

creation\_time = datetime.strptime(image['CreationDate'], '%Y-%m-%dT%H:%M:%S.%fZ')

if (datetime.utcnow() - creation\_time) > timedelta(days=30):

ec2.deregister\_image(ImageId=image['ImageId'])

1. IAM Policy

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"cloudwatch:DeleteAlarms",

"cloudwatch:DescribeAlarmHistory",

"cloudwatch:DescribeAlarms",

"cloudwatch:DescribeAlarmsForMetric",

"cloudwatch:GetMetricStatistics",

"cloudwatch:ListMetrics",

"sqs:SendMessage",

"cloudwatch:PutMetricAlarm",

"ec2:ModifySnapshotAttribute",

"ec2:ResetSnapshotAttribute",

"ec2:Describe\*",

"ec2:DescribeInstances",

"ec2:Create\*",

"ec2:DeregisterImage",

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents",

"iam:GetRole",

"iam:ListRoles",

"lambda:\*"

],

"Resource": "\*",

"Effect": "Allow"

}

]

}