

AWS EKS Cost-Optimization: Lambda, Karpenter, EBS, S3

Check GitHub for helpful DevOps tools:

Michael Robotics

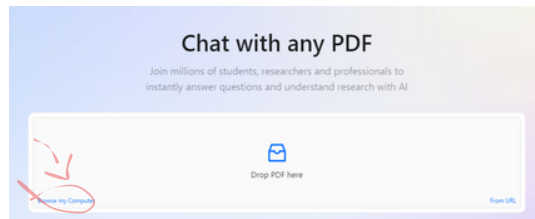
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

 <https://github.com/MichaelRobotics>



Ask Personal AI Document assistant to learn
interactively (FASTER)!

- 1 Download PDF
 - 2 Go to website
 - 3 Browse file
 - 4 Chat with Document
- 1 <https://github.com/MichaelRobotics/DevOpsTools/blob/main/AWSopt.pdf>
- 2 | Click there to go to ChatPdf website
- 3
- 4 Ask questions about document!




Completely new to AWS and Kubernetes?

If you are completely new to this topic, using a document assistant to understand the many definitions can be helpful. However, the best way to start is by watching this video, which I believe provides the best explanation for beginners starting their journey with Kubernetes

Kubernetes Crash Course for Absolute Beginners

Hands-On Kubernetes Tutorial | Learn Kubernetes in 1 Hour - Kubernetes Course for Beginners

 https://www.youtube.com/watch?v=s_o8dwzRlu4&ab_channel=TechWorldwithNana



Essential for this PDF is a thorough knowledge of networking. I highly recommend A Cloud Guru, which offers beginner-friendly courses like the AWS Certified Cloud Practitioner and hands-on labs. As you advance, explore certifications such as AWS Solutions Architect or DevOps Engineer to deepen your skills.

CloudGuru

Driven by a mission to teach the world to cloud, A Cloud Guru, a Pluralsight Company, helps teams level up their AWS skills, prepare for certification exams

<https://www.pluralsight.com/cloud-guru/pricing>



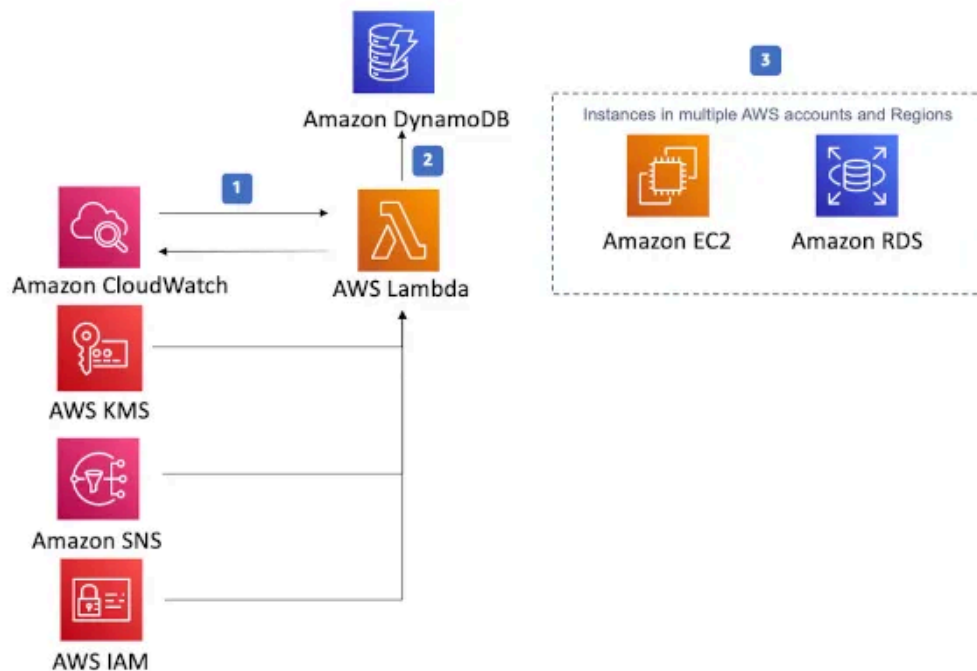
What is AWS Cost-Optimization?

Cost optimization involves analyzing and adjusting resource usage to ensure you only pay for necessary resources, balancing cost with performance. The goal is to eliminate waste and maximize return on investment.



How AWS Cost-Optimization works?

AWS cost optimization works by using tools like AWS Cost Explorer for visibility, right-sizing instances based on usage, and employing auto-scaling to adjust resources dynamically. Regular audits of underutilized resources and strategic choices in regions and storage help align cloud spending with business needs, avoiding unnecessary costs.



AWS Cost-Optimization: Why and When

AWS cost optimization is crucial for reducing unnecessary spending, improving operational efficiency, and maximizing the value of cloud investments. It ensures that businesses align their cloud expenses with actual usage, avoiding over-provisioning and capitalizing on discounts like Savings Plans.

A typical use case for AWS cost optimization involves a company running several idle EC2 instances or over-provisioned storage, leading to unnecessary costs. By using tools like AWS Cost Explorer and AWS Compute Optimizer, the company can right-size instances, remove underutilized resources, and implement savings plans to cut costs while maintaining performance.

System Requirements

- AWS account

If you want to install it on a different cloud provider, ask in the comments and I'll provide a solution for you!

AWS EKS Cost-Optimization: Main components & packages

- Lambda functions
- EKS
- EBS
- Karpenter
- S3



AWS EKS Cost-Optimization: Delete Stale EBS snapshots using Lambda & S3 Lifecycle

1) Understand popular case study

A large financial services company running on AWS faced storage cost inefficiencies due to unmanaged EBS snapshots. When EC2 instances were terminated, the associated EBS volumes were deleted, but the snapshots, which are backup points, remained in their AWS environment.

You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:

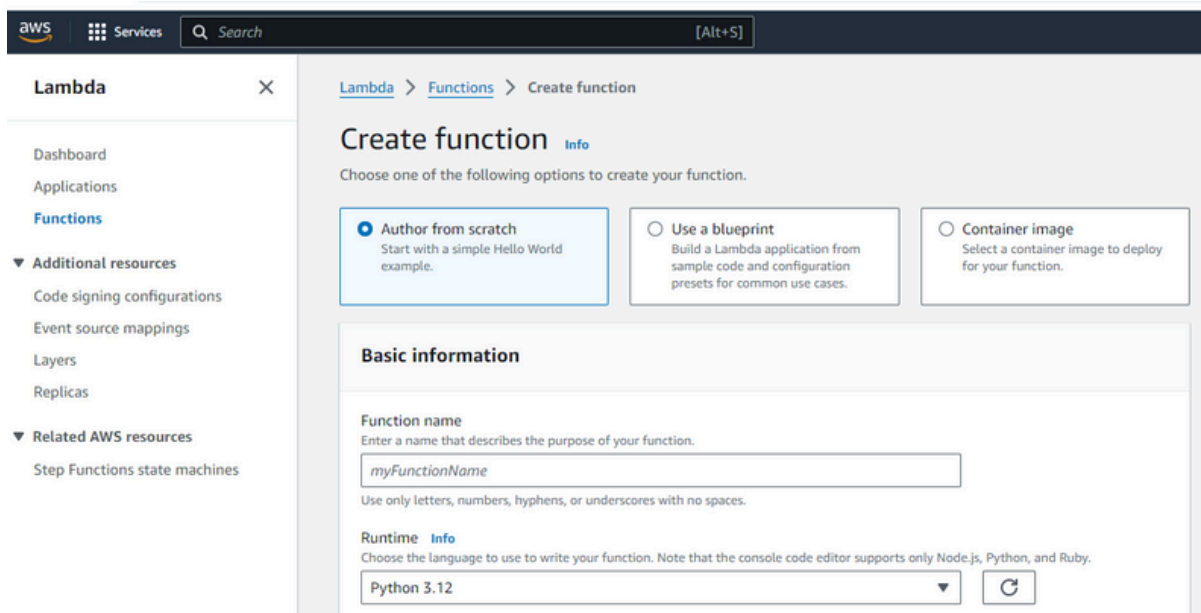
| | | | | | |
|---|---|-------------------------------------|---|---------------------------------|----|
| Instances (running) | 1 | Auto Scaling Groups | 0 | Dedicated Hosts | 0 |
| Elastic IPs | 0 | Instances | 1 | Key pairs | 11 |
| Load balancers | 0 | Placement groups | 0 | Security groups | 16 |
| Snapshots  | 1 | Volumes | 1 | | |

2) Solution

The company automated the deletion of orphaned EBS snapshots by using AWS Lambda, which was triggered by CloudWatch Events when EC2 instances were terminated. Lambda identified and deleted the snapshots, while IAM roles ensured secure access with necessary permissions.

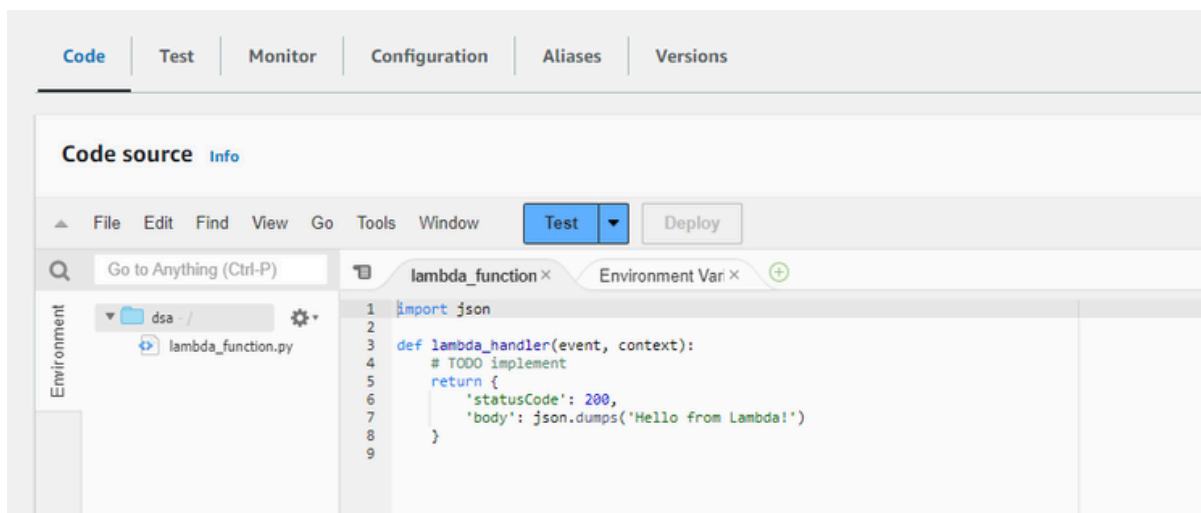
3) Create Lambda function

Navigate towards AWS and create Lambda



The screenshot shows the AWS Lambda 'Create function' page. The left sidebar contains the 'Lambda' menu with options like Dashboard, Applications, Functions, and Additional resources. The main content area is titled 'Create function' and includes a breadcrumb 'Lambda > Functions > Create function'. Below the title, there are three radio buttons for creating a function: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. The 'Basic information' section contains a 'Function name' field with the value 'myFunctionName' and a 'Runtime' dropdown menu set to 'Python 3.12'.

You can manually test your Lambda code in the AWS console before setting up a CloudWatch event to ensure it functions work correctly. Below is the Lambda code in Python:



The screenshot shows the AWS Lambda 'Code source' page. The top navigation bar includes tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The 'Code source' tab is active, showing a code editor with the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

4) Lambda cost-optimization

Keep in mind that Lambda functions have a standard execution time of 3 seconds, and longer execution times will increase costs

The screenshot shows the AWS Lambda console configuration page. It includes sections for Memory (128 MB), Ephemeral storage (512 MB), SnapStart (None), Timeout (0 min 5 sec), and Execution role (Use an existing role). The SnapStart section mentions supported runtimes: Java 11, Java 17, Java 21.

5) Write Lambda function

Create object to interact with EC2, create list of all EBS snapshots owned by account and get information about all EC2 in running state.

```
1 import boto3
2
3 def lambda_handler(event, context):
4     ec2 = boto3.client('ec2')
5
6     # Get all EBS snapshots
7     response = ec2.describe_snapshots(OwnerIds=['self'])
8
9     # Get all active EC2 instance IDs
10    instances_response = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
11    active_instance_ids = set()
```


Extract the InstanceIds of running instances and store it in active_instance_ids.

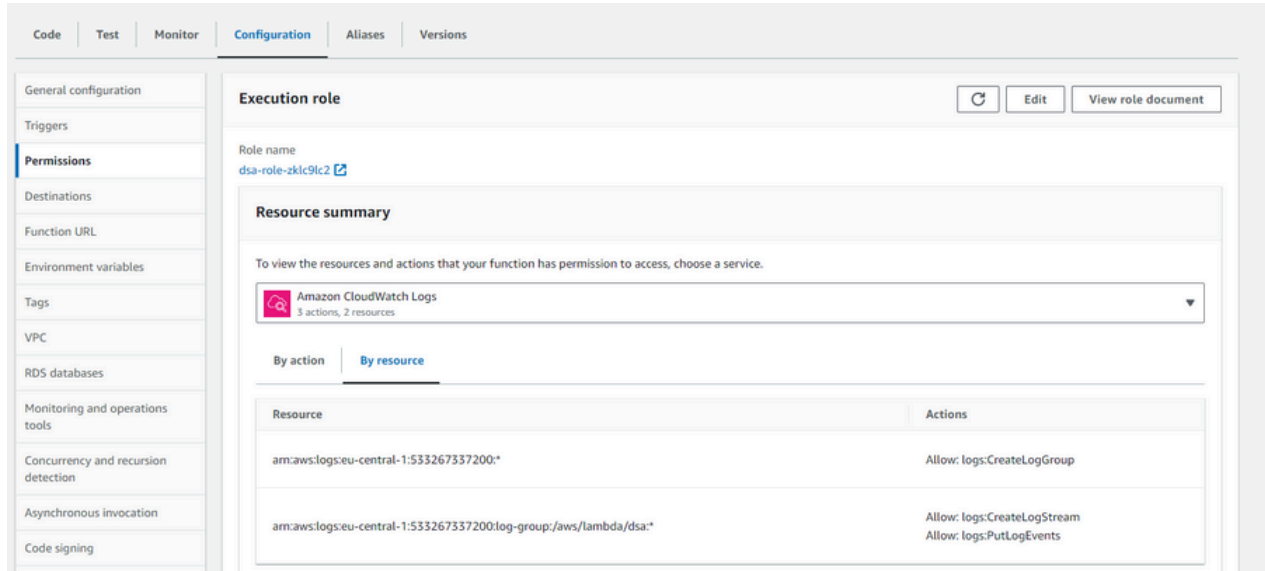
```
for reservation in instances_response['Reservations']:
    for instance in reservation['Instances']:
        active_instance_ids.add(instance['InstanceId'])
```

Iterate through all snapshots and volumes. Case 1: Delete snapshots not associated with any volume. Case 2: Delete snapshots associated with a volume if the volume's EC2 instance does not exist. Case 3: Delete snapshots linked to volumes if the volume has already been deleted.

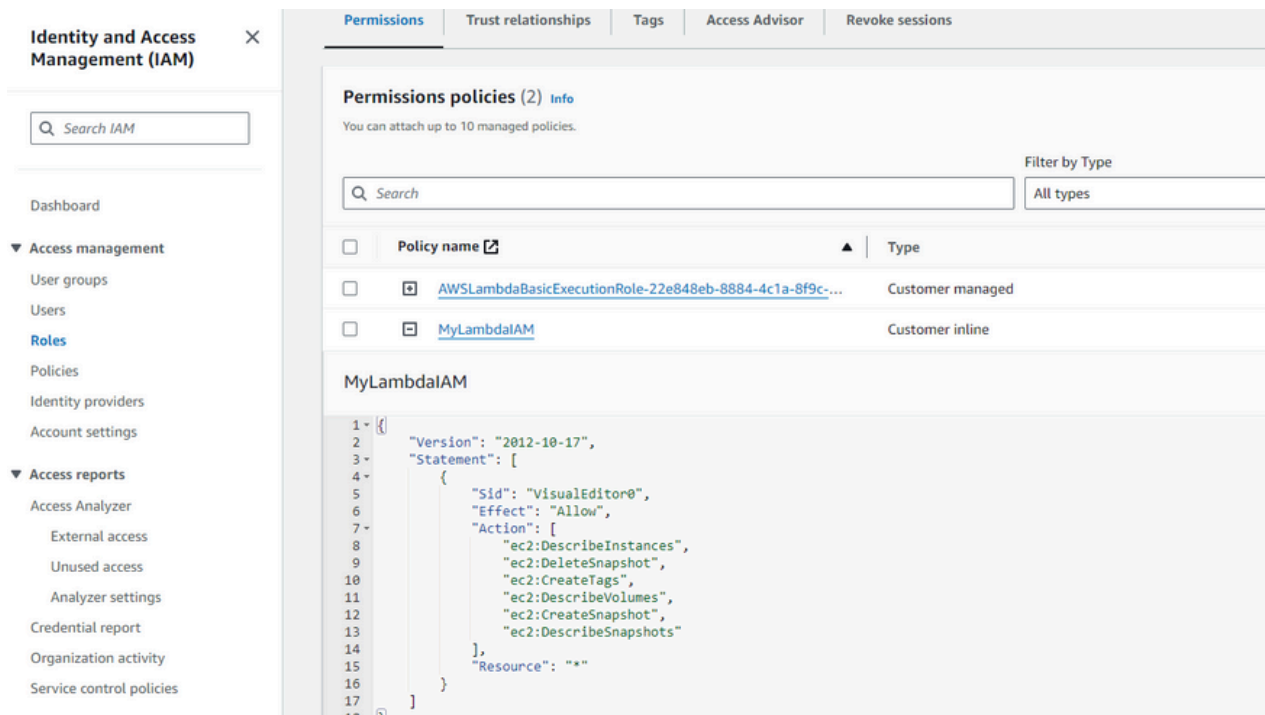
```
17 # Iterate through each snapshot and delete if it's not attached to any volume or the volume is not attached to a running instance
18 for snapshot in response['Snapshots']:
19     snapshot_id = snapshot['SnapshotId']
20     volume_id = snapshot.get('VolumeId')
21
22     if not volume_id:
23         # Delete the snapshot if it's not attached to any volume
24         ec2.delete_snapshot(SnapshotId=snapshot_id)
25         print(f"Deleted EBS snapshot {snapshot_id} as it was not attached to any volume.")
26     else:
27         # Check if the volume still exists
28         try:
29             volume_response = ec2.describe_volumes(VolumeIds=[volume_id])
30             if not volume_response['Volumes'][0]['Attachments']:
31                 ec2.delete_snapshot(SnapshotId=snapshot_id)
32                 print(f"Deleted EBS snapshot {snapshot_id} as it was taken from a volume not attached to any running instance.")
33         except ec2.exceptions.ClientError as e:
34             if e.response['Error']['Code'] == 'InvalidVolume.NotFound':
35                 # The volume associated with the snapshot is not found (it might have been deleted)
36                 ec2.delete_snapshot(SnapshotId=snapshot_id)
37                 print(f"Deleted EBS snapshot {snapshot_id} as its associated volume was not found.")
```

4) Write IAM for lambda function

Lambda function needs specific IAM permissions to interact with EC2 instances

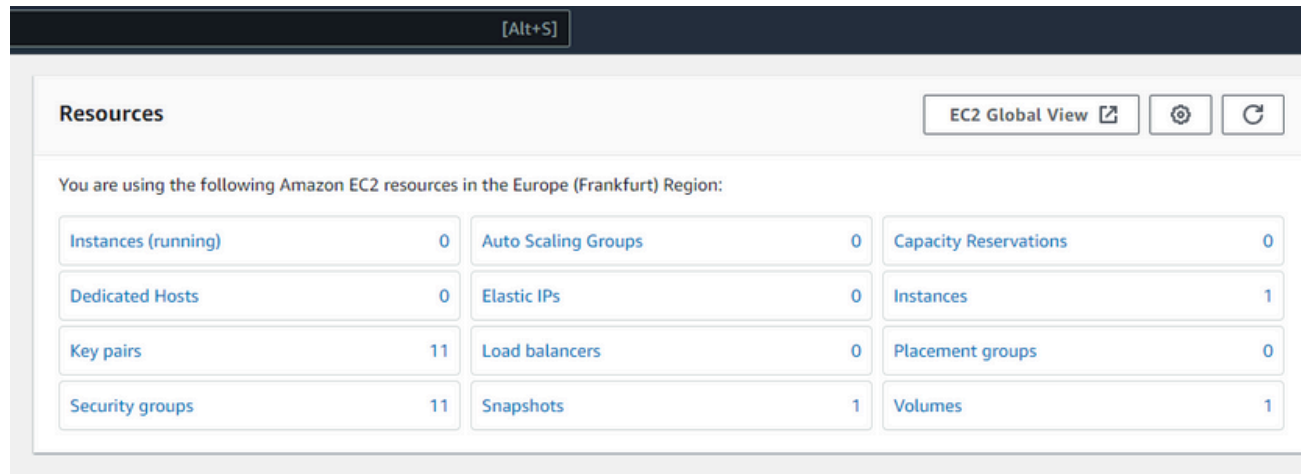


Create new custom IAM role with those policies:



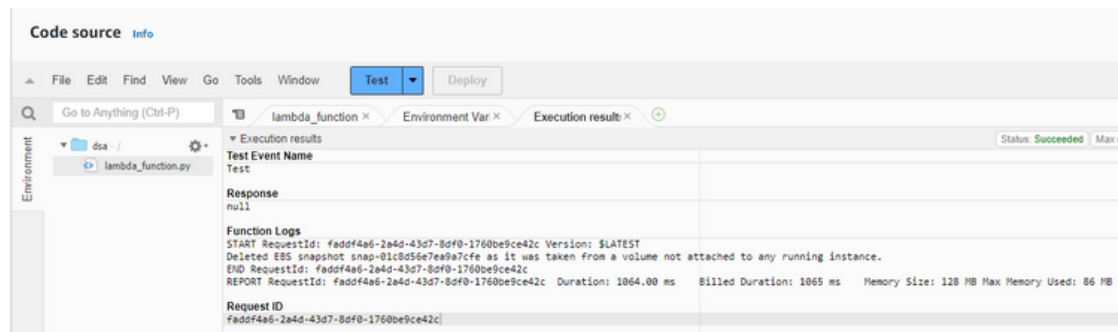
5) Test Lambda function by clicking deploy

Check EC2 Dashboard before running lambda function



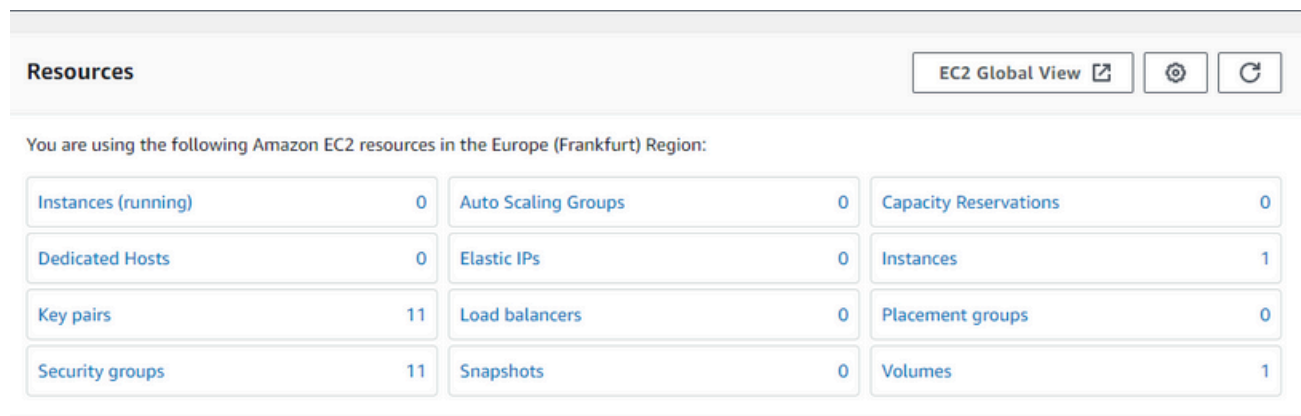
| Resources | | |
|--|----|-----------------------|
| You are using the following Amazon EC2 resources in the Europe (Frankfurt) Region: | | |
| Instances (running) | 0 | Auto Scaling Groups |
| Dedicated Hosts | 0 | Elastic IPs |
| Key pairs | 11 | Load balancers |
| Security groups | 11 | Snapshots |
| | | Capacity Reservations |
| | | Instances |
| | | Placement groups |
| | | Volumes |

Run Lambda function (Click Deploy) - Process should run without errors



| Code source | Info |
|-------------------------------------|---|
| File Edit Find View Go Tools Window | Test Deploy |
| Go to Anything (Ctrl-P) | lambda_function x Environment Var x Execution results x |
| Environment | Execution results Status: Succeeded Max r |
| dsa - / | Test Event Name |
| lambda_function.py | Test |
| | Response |
| | null |
| | Function Logs |
| | START RequestId: faddf4a6-2a4d-43d7-8df0-1760be9ce42c Version: \$LATEST |
| | Deleted EBS snapshot snap-01c8d56e7ea9a7cfe as it was taken from a volume not attached to any running instance. |
| | END RequestId: faddf4a6-2a4d-43d7-8df0-1760be9ce42c |
| | REPORT RequestId: faddf4a6-2a4d-43d7-8df0-1760be9ce42c Duration: 1064.00 ms Billed Duration: 1065 ms Memory Size: 128 MB Max Memory Used: 86 MB |
| | Request ID |
| | faddf4a6-2a4d-43d7-8df0-1760be9ce42c |

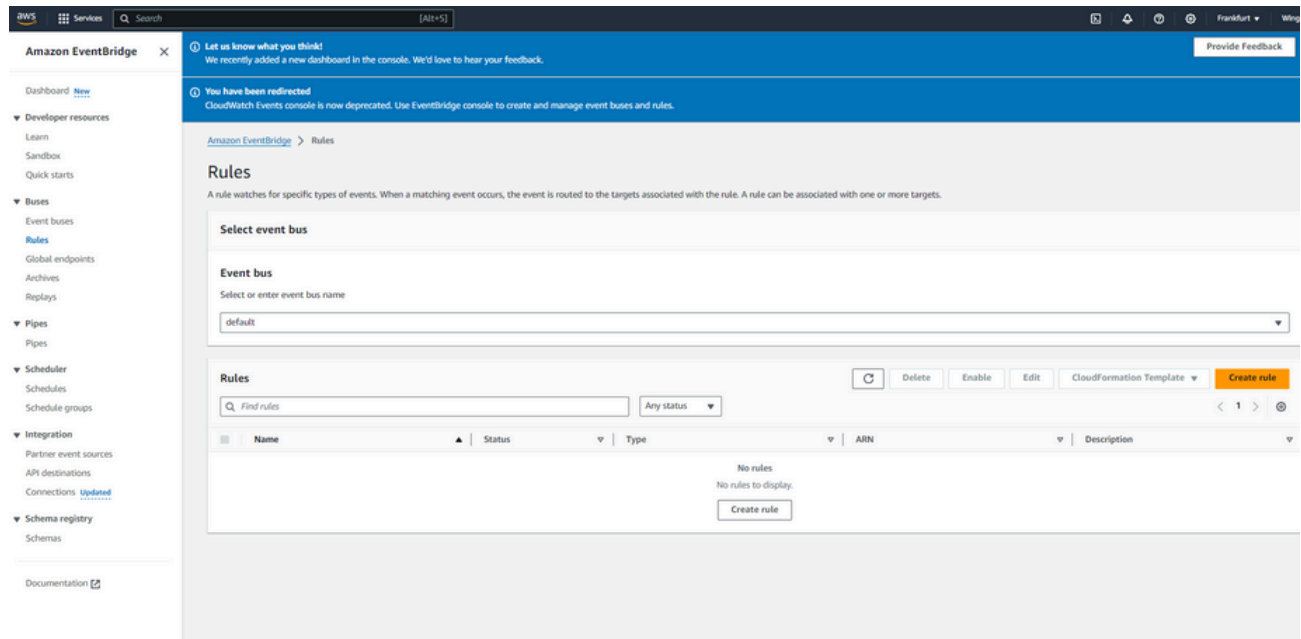
Check EC2 Dashboard, Snapshot counter should be 0



| Resources | | |
|--|----|-----------------------|
| You are using the following Amazon EC2 resources in the Europe (Frankfurt) Region: | | |
| Instances (running) | 0 | Auto Scaling Groups |
| Dedicated Hosts | 0 | Elastic IPs |
| Key pairs | 11 | Load balancers |
| Security groups | 11 | Snapshots |
| | | Capacity Reservations |
| | | Instances |
| | | Placement groups |
| | | Volumes |

6) Configure Event Trigger using CloudWatch

Navigate to CloudWatch -> Rules. You will get redirected to Amazon EventBridge. **Create rule**



Continue in EventBridge Scheduler and follow instructions. Remember, more Lambda execution time equals more costs!

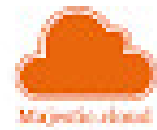
7) Learn cost-optimization techniques with S3 Lifecycle Rules

Check out this fantastic short video that explains the topic perfectly!

How to set up S3 Lifecycle Rules to save on S3 costs

AWS S3 Lifecycle Rules is a powerful feature that automates the process of managing your S3 objects

https://www.youtube.com/watch?v=Wilow2AECho&ab_channel=Majestic.cloud



AWS EKS Cost-Optimization: Karpenter Intro

1) How standard kubernetes Auto Scaling works

When traffic increases and the EC2 instance where the node is deployed has horizontal autoscaling set up but lacks available capacity, new pods cannot be deployed, causing them to enter an unschedulable state. At this point, the Cluster Autoscaler is triggered to scale up the nodes. Karpenter is designed as a specific use case to replace the Cluster Autoscaler for more efficient scaling

2) Why Karpenter is better

Cluster Autoscaler first triggers the Auto Scaling Group (ASG) to launch new instances before adding them to the node group, which can introduce delays. Karpenter, on the other hand, directly provisions nodes without needing to involve the ASG, making the scaling process faster.

While Cluster Autoscaler requires each node group to be tied to a specific EC2 instance type, Karpenter allows node groups to use any EC2 instance type. Additionally, Karpenter intelligently selects the optimal EC2 instance type for pod deployment based on cost efficiency, simplifies upgrades and patching, and supports a wide range of workloads, including machine learning (ML) and generative AI."

4) Karpenter use 2 files: NodePool and EC2NodeClass

A NodePool in Karpenter is a group of dynamically managed EC2 instances that Karpenter provisions to run Kubernetes pods. It defines the characteristics and rules for creating and managing worker nodes, ensuring efficient scaling based on workload demands.

```
# Operators { In, NotIn, Exists, DoesNotExist, Gt, and Lt }
# https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-selector
requirements:
  - key: "karpenter.k8s.aws/instance-category"
    operator: In
    values: ["c", "m", "r"]
    # minValues here enforces the scheduler to consider at least this many values
    # This field is ALPHA and can be dropped or replaced with a more specific requirement
    minValues: 2
  - key: "karpenter.k8s.aws/instance-family"
    operator: In
    values: ["m5", "m5d", "c5", "c5d", "c4", "r4"]
    minValues: 5
```

The EC2NodeClass in Karpenter defines the configuration and behavior of EC2 instances used to fulfill a NodePool's resource needs, similar to a "Launch Template" in an EC2 Auto Scaling Group. Key components of an EC2NodeClass are: AMI, Security Group, Subnet.

```
status:
  # Resolved subnets
  subnets:
    - id: subnet-0a462d98193ff9fac
      zone: us-east-2b
    - id: subnet-0322dfafd76a609b6
      zone: us-east-2c
    - id: subnet-0727ef01daf4ac9fe
      zone: us-east-2b
    - id: subnet-00c99aeafe2a70304
```

```
securityGroups:
  - id: sg-041513b454818610b
    name: ClusterSharedNodeSecurityGroup
  - id: sg-0286715698b894bca
    name: ControlPlaneSecurityGroup-1AQ073TSAAPW

# Resolved AMIs
amis:
  - id: ami-01234567890123456
    name: custom-ami-amd64
```

4) Scheduling workloads with Karpenter

If your pods have no requirements for how or where to run, you can let Karpenter choose nodes from the full range of available cloud provider resources. However, by taking advantage of Karpenter's model of layered constraints, you can be sure that the precise type and amount of resources needed are available to your pods.

Constraints you can request include:

- **Resource requests:** Request that certain amount of memory or CPU be available.
- **Node selection:** Choose to run on a node that has a particular label (nodeSelector).
- **Node affinity:** Draws a pod to run on nodes with particular attributes (affinity).
- **Topology spread:** Use topology spread to help ensure availability of the application.
- **Pod affinity/anti-affinity:** Draws pods towards or away from topology domains based on the scheduling of other pods.

5) Consolidation

Karpenter reduce cluster cost by removing empty nodes, moving pods to underutilized nodes and replacing nodes with cheaper variants.

6) Demo

Karpenter offers a great demo project, which even includes the ability to implement Grafana for monitoring

Getting Started with Karpenter

Set up a cluster and add Karpenter

<https://karpenter.sh/v0.37/getting-started/getting-started-with-karpenter/>



Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.



<https://github.com/MichaelRobotics>



PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!