BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

First Semester 2025- 26

SSWT ZG628T DISSERTATION

MID SEMESTER EVALUATION FORM

ID No .**2021wa15838** Name of Student: **Ajad Ramjit Chauhan**

Name of Supervisor: Shafique Khan

Name of the Examiner(s):- GANESH PAWADE

Dissertation Title: **Intelligent AWS EC2 Cost Optimization and Stale Resource Management**

Section II
(To be filled by the Supervisor in consultation with the examiner(s))

Comments on the dissertation from Examiner and Supervisor (Select Y or N)

1. **Quantum of work**
   - ✓ **Justifiable as efforts for 8 weeks duration** [✓ Y] / N
   - ✓ **Work is in line with the commitments made in outline** [✓ Y] / N
2. **Type of work**
   - ✓ **Client assignment** Y / [✓ N]
   - ✓ **Organization specific task** [✓ Y] / N
   - ✓ **General study project such as white paper** Y / [✓ N]
   - ✓ **Any other (kindly elaborate below in a line or two if Y** Y / [✓ N]

3. **Nature of work**
   - ✓ **Routine in nature** [✓ Y] / N
   - ✓ **Involved creativity and rational thinking** [✓ Y] / N

   Kindly elaborate below if answer for above is “Y”

4. **Evaluation methodology**
   - ✓ **Evaluation done based on presentation to supervisor and examiner** [✓ Y] / N
   - ✓ **Evaluation done through Viva conducted by supervisor and examiner** [✓ Y] / N
   - ✓ **Student regularly interacted with supervisor and incorporated the suggestions made** [✓ Y] / N

- ✓ **Brief description on the report submitted, quality of presentation and suggestions given for improvement**

5. **Mid semester evaluation matrix**

Dimension Rank→	1	2	3	4	5
Student abilities in general					
Understanding of the subject of dissertation					✓
Creative thinking ability to come up with new ideas				✓	
Viva / Seminar presentation					
Communication ability				✓	
Organization of material				✓	
Response to review questions				✓	
Cohesive thinking ability			✓		
Report submitted					
Report structure and format				✓	
Technical content of the report			✓		
Explanation on the significance of the assignment				✓	
Analysis of alternative approaches			✓		

Any other comments:No

Pawade. Ganesh.

S. Ahmed

Date: 21/09/2025 Signature of examiner(s) Signature of Supervisor

1. INTRODUCTION

1.1 Background

Cloud has made it incredibly easy to launch what we need, when we need it—and that's its greatest strength and its biggest trap. In day-to-day work, development or test EC2 instances stay on long after a sprint ends, and storage such as EBS volumes or snapshots outlives the project that created them. Over time, these "quiet" resources add up to real spend, even when they deliver no value.

1.2 Problem Statement

Manually chasing idle instances and forgotten storage doesn't scale. It's slow, error-prone, and distracting for teams whose focus should be delivery. Without automated detection and a simple way to act on waste, organizations leave savings on the table and see bills rise without a clear explanation.

1.3 Purpose and Objectives

This dissertation develops an automated, AWS-native system that finds waste early and makes it easy to fix safely. It will:

- Analyze EC2 utilization to surface idle instances that are candidates for schedule, stop, or rightsizing.
- Detect stale resources—including unattached EBS volumes, old snapshots, and unused Elastic IPs—that quietly incur charges.
- Automate cleanup using serverless functions with approvals and guardrails.
- Visualize costs, findings, and realized savings through an interactive dashboard.
- Establish lightweight rules and tagging practices for ongoing cost governance.

1.4 Scope

The project delivers a working prototype in a single AWS account, focused on EC2, EBS, Elastic IPs, and Snapshots. It includes: (a) detection logic and automation workflows, and (b) a self-service dashboard for visibility and approval-based actions. The outcome is a practical, low-overhead approach that teams can adopt without changing how they ship software.

2.SYSTEM ANALYSIS & DESIGN

2.1. Proposed System

The proposed system is a cloud-native, event-driven application that operates on a schedule. It will:

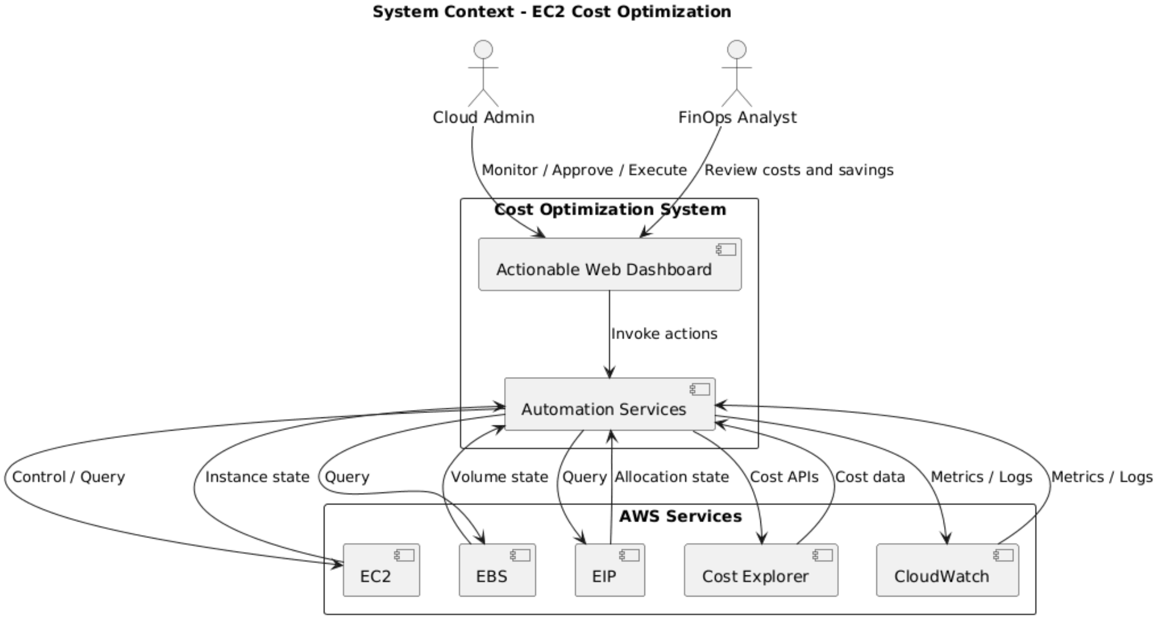
1. **Monitor:** Continuously collect EC2 system metrics (CPU, Network, Disk) using the CloudWatch Agent.
2. **Analyze:** Use AWS Lambda functions to query these metrics and apply algorithms to identify idle instances.
3. **Detect:** Use Lambda functions to scan the AWS environment for resources that meet the criteria for being "stale."
4. **Act:** Automate the stopping of idle instances and deletion of stale resources, with safeguards.
5. **Visualize:** Present all findings, cost data, and historical trends in a Streamlit dashboard for user insight and manual override.

2.2. Technology Stack & Justification

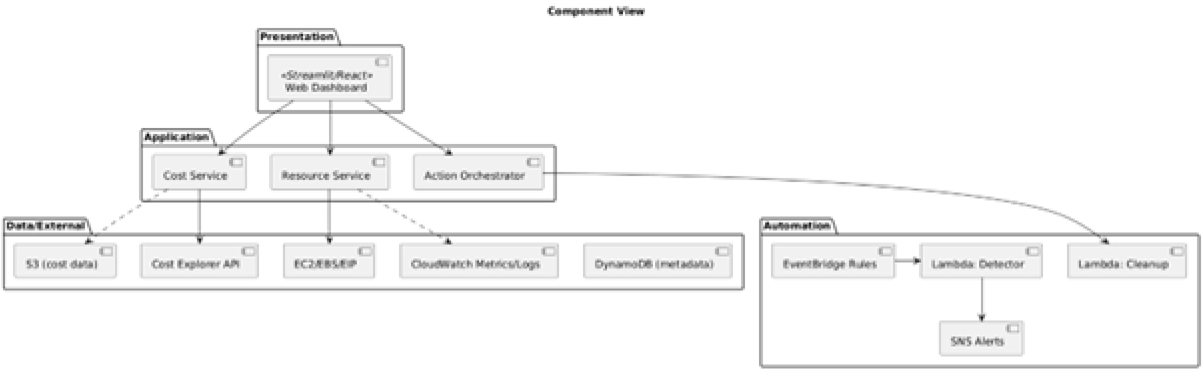
Technology	Justification
AWS Lambda	Serverless compute for event-driven, pay-per-use automation. Eliminates need to manage servers.
Amazon CloudWatch	Centralized monitoring and metrics repository. Provides data for idle analysis and triggers functions.
AWS Cost Explorer API	Provides granular access to cost and usage data for visualization and reporting.
AWS CLI / Boto3	The primary tools for programmatically interacting with AWS services to describe and manage resources.
Python	The chosen runtime for Lambda functions and the Streamlit dashboard due to its rich SDK (Boto3) and data libraries.
Streamlit	A rapid application development framework for building interactive data dashboards in pure Python.
IAM	AWS Identity and Access Management is critical for enforcing the principle of least privilege and securing the system.

3.SYSTEM Architecture

The system is designed with a serverless, event-driven architecture for maximum cost-efficiency and scalability.



3.1 Architecture & Implementation

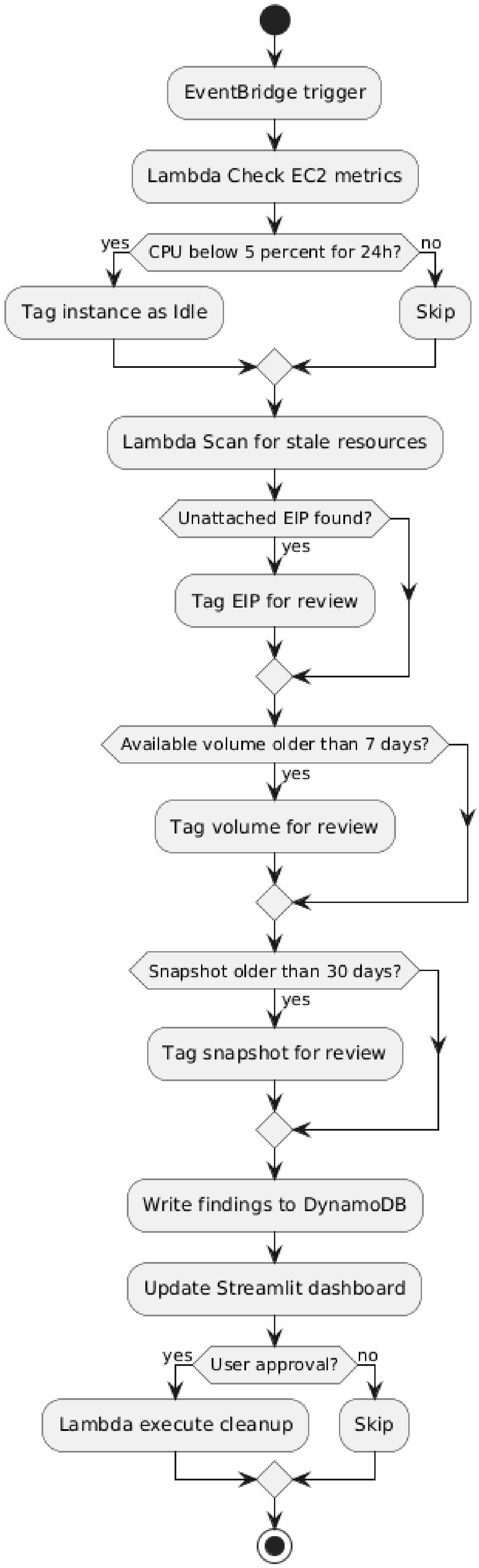


3.2 Artitechture component and its purpose

Component	**Purpose**	**Free Tier Impact**
EC2 t2.micro	Primary compute resource	750 hours/month included
CloudWatch Agent	System metrics collection	Basic monitoring free
Lambda Functions	Automated cleanup & control	1M requests + 400K GB-s free
SNS	Alert notifications	1000 notifications/month free
S3/DynamoDB	Cost data storage	5GB storage free
EventBridge	Scheduled automation	Free for AWS service targets
Streamlit Dashboard	Cost visualization interface	Run locally or minimal EC2 usage

3.3 Flow diagram

Idle and Stale Resource Workflow (Simple)



4. WEEKLY PROGRESS SUMMARY (Weeks 1-9)

Weeks 1-2: Research, Setup, and Requirements

What was accomplished

- Examined core references—the AWS Well-Architected Framework, serverless design patterns, and recent work on cloud cost optimization—to ground design choices in proven practice.
- Captured clear functional and non-functional requirements, focusing on accuracy of detection, safety (approval-based cleanup), and low operating cost.
- Established a secured AWS Free Tier environment for development and testing, with billing alerts enabled.
- Implemented least-privilege IAM policies to protect actions like cost queries and resource cleanup.
- Prepared a productive local toolchain (AWS CLI, Python, VS Code, Git) for rapid, reproducible iteration.

Challenges and how they were addressed

- Scope realism: Balanced ambition with the dissertation timeline by prioritizing high-impact features first (idle EC2 detection, stale resource signals, approval-based cleanup), deferring nice-to-have items.
- Service selection: Narrowed the AWS surface area to EC2, CloudWatch, Lambda, EventBridge, SNS, S3, DynamoDB, and Cost Explorer—services that deliver the most value for monitoring, automation, and cost insights.

Troubleshooting readiness
A practical runbook is in place for the most common early issues:

- CloudWatch Agent setup and metric flow verification.
- Lambda timeout/memory tuning and idempotency patterns.
- Cost Explorer permission and region constraints for API access.
- Streamlit dashboard connectivity and port/security-group checks.

Key learnings

- Security is foundational: least-privilege IAM and guardrails around cleanup actions build trust and reduce risk.

- The AWS shared responsibility model informs design decisions—from data handling to credential scope.
- Serverless services (Lambda, EventBridge, SNS) reduce operational overhead and cost compared to always-on servers, making them ideal for this use case.

✚ Status and deliverables

- **Status: Completed on schedule.**
- Deliverables: Literature Review, Secured AWS Account (Free Tier) with billing alerts, and a Functional Requirements Specification (FRS) aligned to the dissertation plan.

Weeks 3–4: EC2 Launch and Monitoring and metrics collection.

- Stood up a t2.micro on Amazon Linux 2 and attached a purpose-built IAM role—tight permissions for CloudWatch and EC2 only.
- Wrote a minimal IAM policy (for example, cloudwatch:PutMetricData, ec2:DescribeTags) to keep blast radius small.
- Installed the unified CloudWatch Agent and shipped a custom config.json to capture memory, disk, and network—signals the default EC2 metrics miss.
- Verified every metric stream in CloudWatch to close the loop from host to dashboard.
- Setting up budget and alarm for threshold CPU utilization reached.

What was tricky—and how it was solved

- Agent config pitfalls: iterated on the JSON until all target metrics appeared with sane units and intervals.
- IAM precision: refined permissions through test-and-tighten cycles to prevent over-granting.

What this phase taught

- Practical IAM craftsmanship (roles, policies, instance profiles).
- AWS SNS to notify alert and alarms.
- The difference between default vs enhanced monitoring—and why the latter matters for right-sizing and idle detection.
- A repeatable recipe for a secure, observable EC2 baseline.

Status: Completed
Deliverables: EC2 t2.micro; CloudWatch Agent installed and configured; CPU, memory, disk, and network collection active.

Weeks 5–6: Load Simulation and Signal Capture

- Chose stress-ng for its fine-grained controls on Amazon Linux; installed and validated.
- Designed a simple but revealing plan: 25% CPU for an hour, 80% CPU for 30 minutes, and multi-hour idle.
- Ran tests, watched the CloudWatch dashboard in real time, and archived metrics for analysis.

Roadblocks and resolutions

- Dialing in utilization on t2.micro: tuned worker counts, timeouts, and sleep cycles to hit target CPU percentages.
- Stability under stress: monitored temperature and I/O saturation, keeping SSH access reliable during peaks.

Why it matters

- Learned how to create predictable load and read the metric story it produces.
- Built a labeled dataset (idle vs active) that will power the idle-detection logic in later automation.

Status: Completed
Deliverables: stress-ng installed and tuned; load plan executed; usage pattern dataset archived.

Week 7: CloudWatch dashboard creation

Activities Completed:

- Designed a logical layout for the CloudWatch dashboard to display key metrics clearly.
- Created a new dashboard named EC2-Optimization-Monitor.
- Added the following widgets:
    - **CPU Utilization:** Line graph showing CPU percentage.
    - **Memory Usage:** Line graph showing mem_used_percent.
    - **Disk I/O:** Line graph showing disk read/write bytes.

  - **Network Traffic:** Line graph showing network packets in/out.

- Configured the time range to 1 day and enabled auto-refresh for real-time monitoring.
- Tested the dashboard during a final load simulation to ensure all widgets populated correctly.

Challenges Faced:

- Selecting the most relevant metrics and statistics (Average, Maximum) for each widget to provide the most insightful view.
- Organizing the dashboard for clarity and ease of use.

Key Learnings:

- The capabilities and limitations of the CloudWatch dashboard service.
- Best practices for cloud infrastructure monitoring and visualization.
- This dashboard serves as the primary debugging tool for the automation logic.

Weeks 8–9: Turning Bills into Insights

- Enabled programmatic Cost Explorer access; built a Python script with Boto3 (get_cost_and_usage) to pull daily costs.
- Flattened the nested API response into a clean Pandas DataFrame and persisted as CSV (S3 integration planned next).
- Spun up a Streamlit app (dashboard.py) to load the CSV, plot daily costs, and support quick visual checks.

Hurdles and how they were cleared

- Complex JSON, simple table: wrote parsing helpers to standardize dates, services, and amounts.
- Local today, S3 tomorrow: designed the storage boundary so switching to S3 is a config change, not a rewrite.

What this unlocks

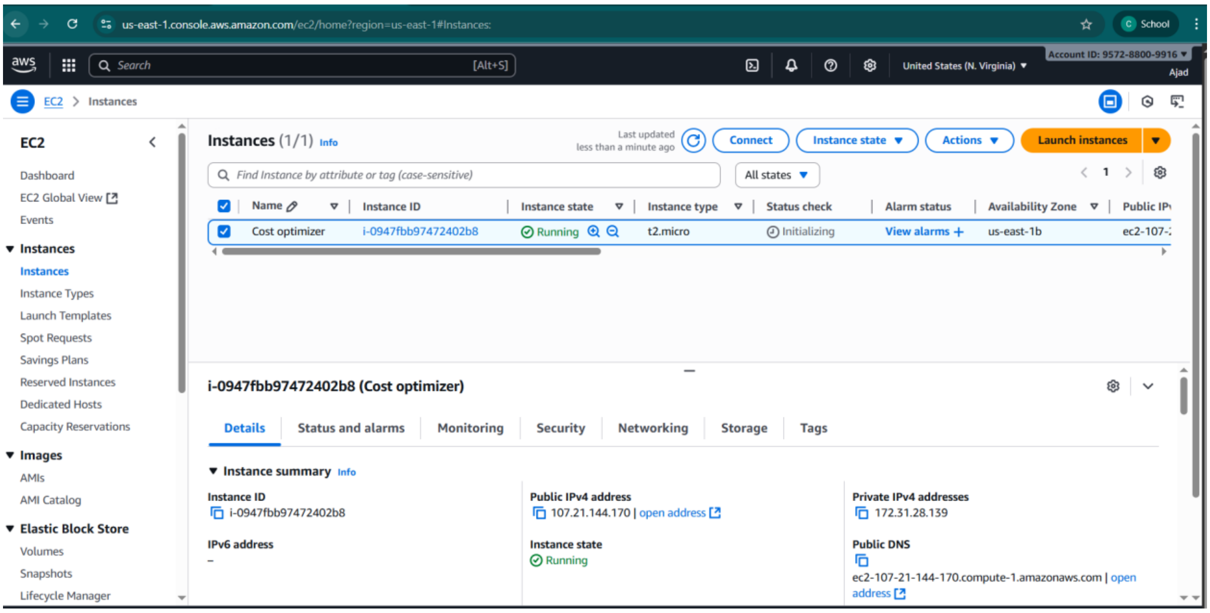
- Comfortable with Cost Explorer APIs and the mechanics of daily cost pipelines.
- Faster feedback loops via Streamlit for spotting anomalies and validating savings over time.

Status: Completed

Deliverables: Cost Explorer enabled; CLI/Python scripts for cost retrieval; local cost store; Streamlit cost dashboard

5. OUTPUT & DELIVERABLES

EC2 instance & IAM setup



Instances (1/1) Info

☑	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
☑	Cost optimizer	i-0947fbb97472402b8	Running	t2.micro	Initializing	View alarms +	us-east-1b	ec2-107-:

i-0947fbb97472402b8 (Cost optimizer)

Instance summary Info

Instance ID: i-0947fbb97472402b8

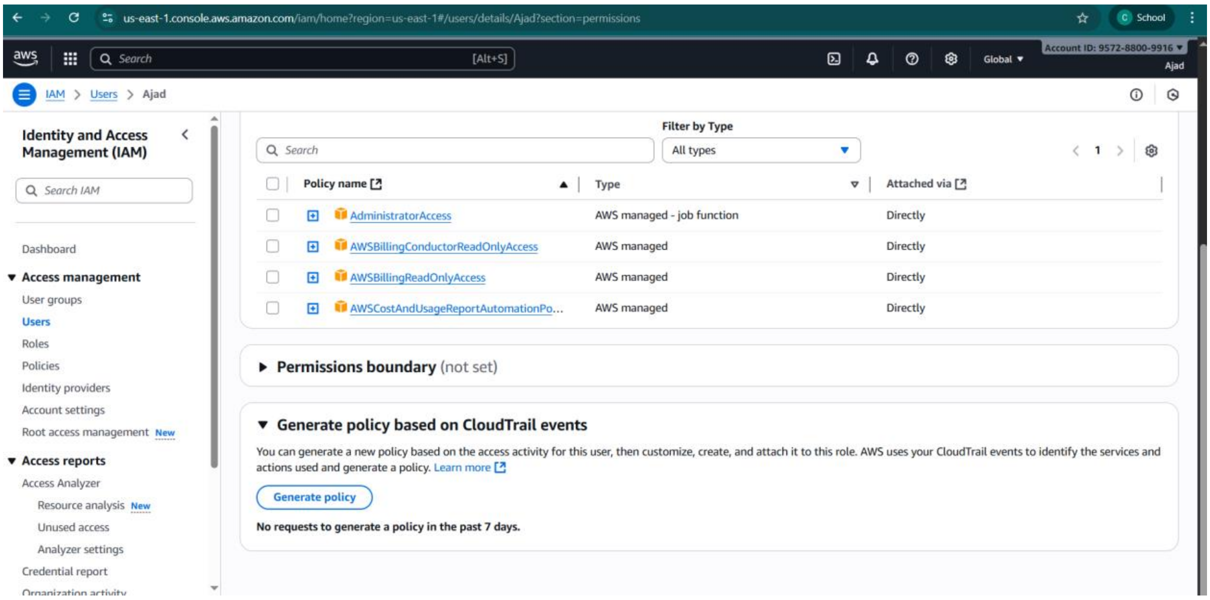
Public IPv4 address: 107.21.144.170 | open address

Private IPv4 addresses: 172.31.28.139

IPv6 address: –

Instance state: Running

Public DNS: ec2-107-21-144-170.compute-1.amazonaws.com | open address



□	Policy name	Type	Attached via
□	AdministratorAccess	AWS managed - job function	Directly
□	AWSBillingConductorReadOnlyAccess	AWS managed	Directly
□	AWSBillingReadOnlyAccess	AWS managed	Directly
□	AWSCostAndUsageReportAutomationPo...	AWS managed	Directly

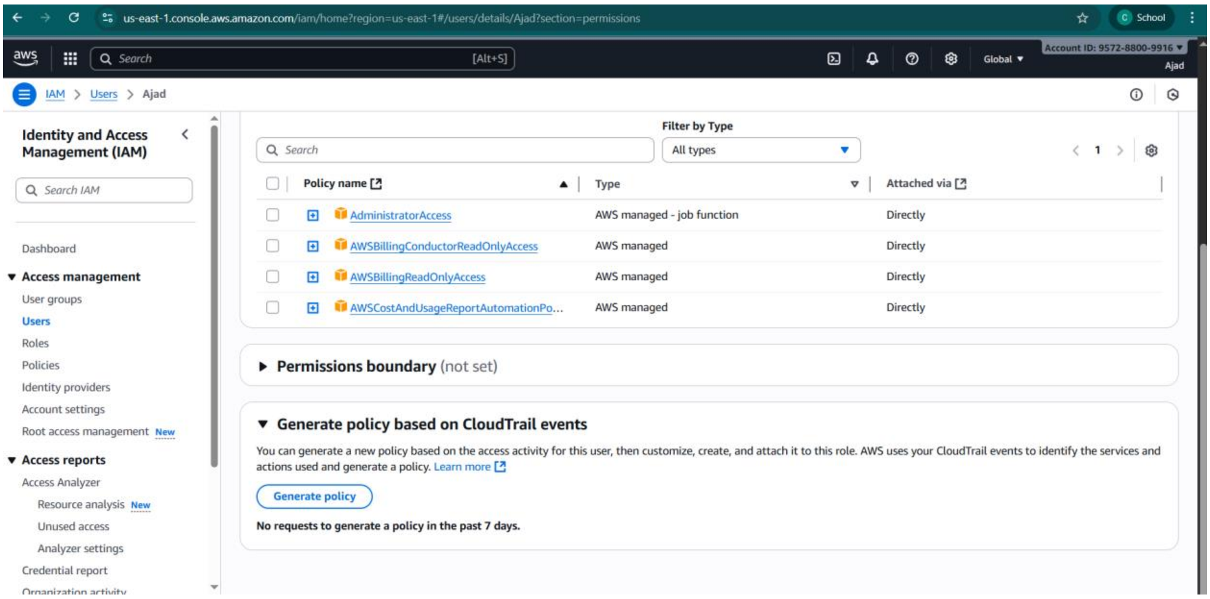
▶ Permissions boundary (not set)

▼ Generate policy based on CloudTrail events

You can generate a new policy based on the access activity for this user, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the services and actions used and generate a policy. Learn more

Generate policy

No requests to generate a policy in the past 7 days.



□	Policy name ▲	Type	Attached via
□	AdministratorAccess	AWS managed - job function	Directly
□	AWSBillingConductorReadOnlyAccess	AWS managed	Directly
□	AWSBillingReadOnlyAccess	AWS managed	Directly
□	AWSCostAndUsageReportAutomationPo...	AWS managed	Directly

▶ **Permissions boundary** (not set)

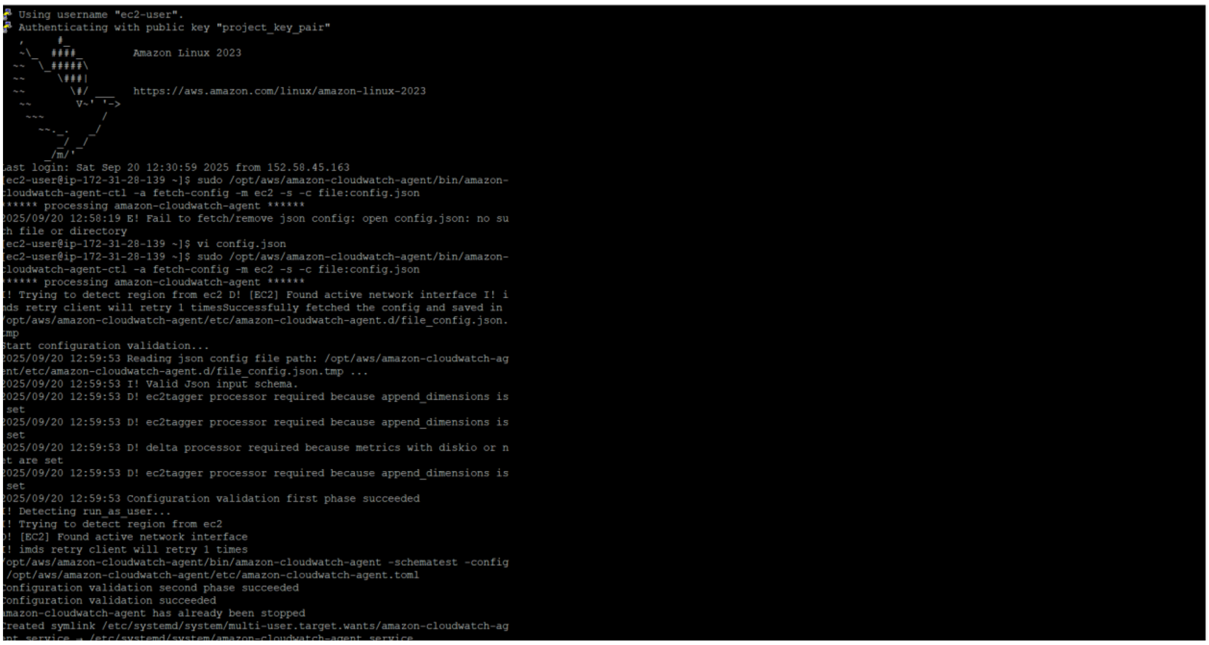
▼ **Generate policy based on CloudTrail events**

You can generate a new policy based on the access activity for this user, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the services and actions used and generate a policy. Learn more

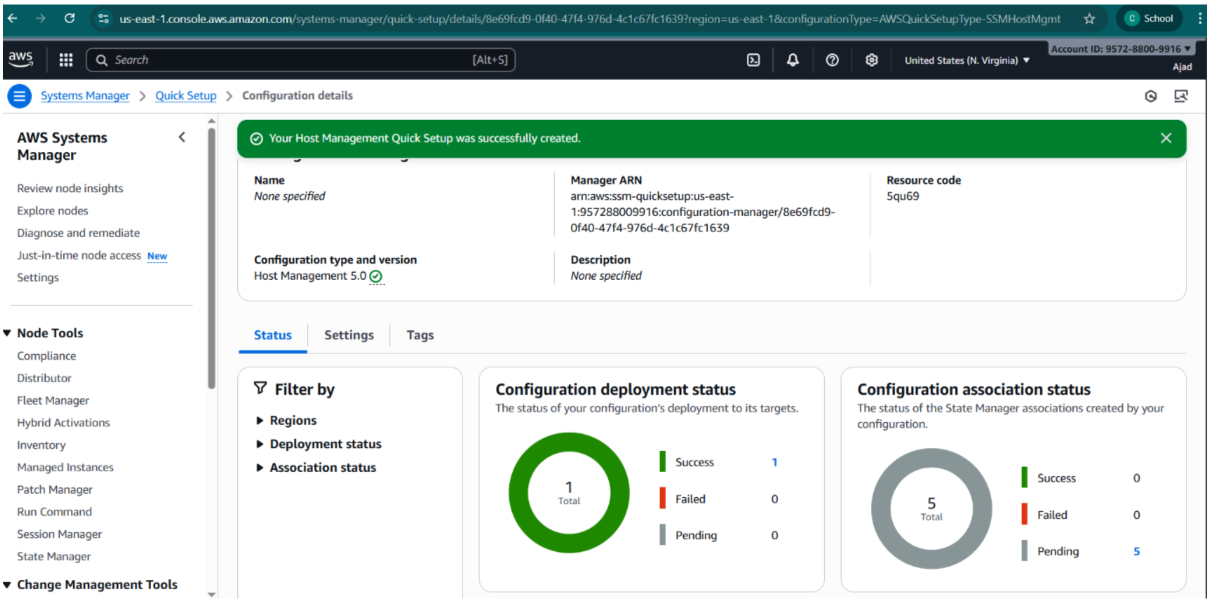
Generate policy

No requests to generate a policy in the past 7 days.

CloudWatch Agent Configuration



```
 Using username "ec2-user".
 Authenticating with public key "project_key_pair"
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sat Sep 20 12:30:59 2025 from 152.58.45.163
[ec2-user@ip-172-31-28-139 ~]$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-
cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:config.json
****** processing amazon-cloudwatch-agent ******
2025/09/20 12:58:19 E! Fail to fetch/remove json config: open config.json: no su
ch file or directory
[ec2-user@ip-172-31-28-139 ~]$ vi config.json
[ec2-user@ip-172-31-28-139 ~]$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-
cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:config.json
****** processing amazon-cloudwatch-agent ******
I! Trying to detect region from ec2 D! [EC2] Found active network interface I! i
mds retry client will retry 1 timesSuccessfully fetched the config and saved in
/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_config.json.
tmp
Start configuration validation...
2025/09/20 12:59:53 Reading json config file path: /opt/aws/amazon-cloudwatch-ag
ent/etc/amazon-cloudwatch-agent.d/file_config.json.tmp ...
2025/09/20 12:59:53 I! Valid Json input schema.
2025/09/20 12:59:53 D! ec2tagger processor required because append_dimensions is
 set
2025/09/20 12:59:53 D! ec2tagger processor required because append_dimensions is
 set
2025/09/20 12:59:53 D! delta processor required because metrics with diskio or n
et are set
2025/09/20 12:59:53 D! ec2tagger processor required because append_dimensions is
 set
2025/09/20 12:59:53 Configuration validation first phase succeeded
I! Detecting run_as_user...
I! Trying to detect region from ec2
D! [EC2] Found active network interface
I! imds retry client will retry 1 times
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -schematest -config
 /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml
Configuration validation second phase succeeded
Configuration validation succeeded
amazon-cloudwatch-agent has already been stopped
Created symlink /etc/systemd/system/multi-user.target.wants/amazon-cloudwatch-ag
ent.service → /etc/systemd/system/amazon-cloudwatch-agent.service
```



us-east-1.console.aws.amazon.com/systems-manager/quick-setup/details/8e69fcd9-0f40-47f4-976d-4c1c67fc1639?region=us-east-1&configurationType=AWSQuickSetupType-SSMHostMgmt
School
Search
[Alt+S]
United States (N. Virginia)
Account ID: 9572-8800-9916
Ajad
Systems Manager > Quick Setup > Configuration details
AWS Systems Manager
Review node insights
Explore nodes
Diagnose and remediate
Just-in-time node access New
Settings
Node Tools
Compliance
Distributor
Fleet Manager
Hybrid Activations
Inventory
Managed Instances
Patch Manager
Run Command
Session Manager
State Manager
Change Management Tools

Your Host Management Quick Setup was successfully created.

Name
None specified

Manager ARN
arn:aws:ssm-quicksetup:us-east-1:957288009916:configuration-manager/8e69fcd9-0f40-47f4-976d-4c1c67fc1639

Resource code
5qu69

Configuration type and version
Host Management 5.0

Description
None specified

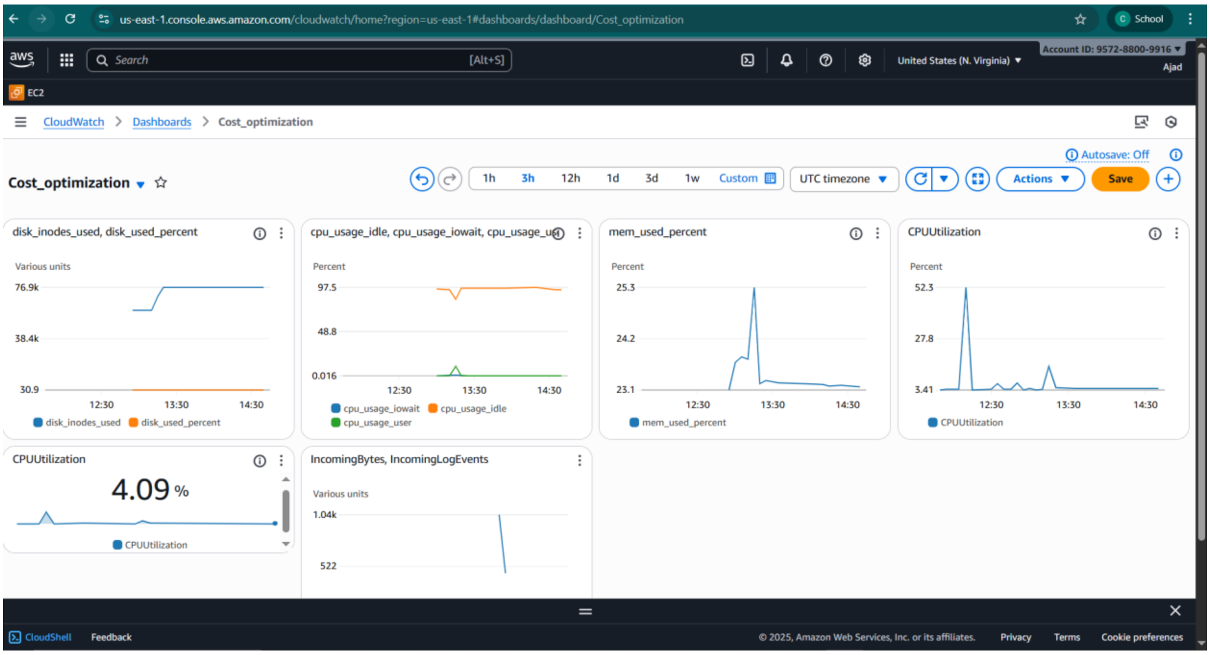
Configuration deployment status
The status of your configuration's deployment to its targets.

1 Total	
Success	1
Failed	0
Pending	0

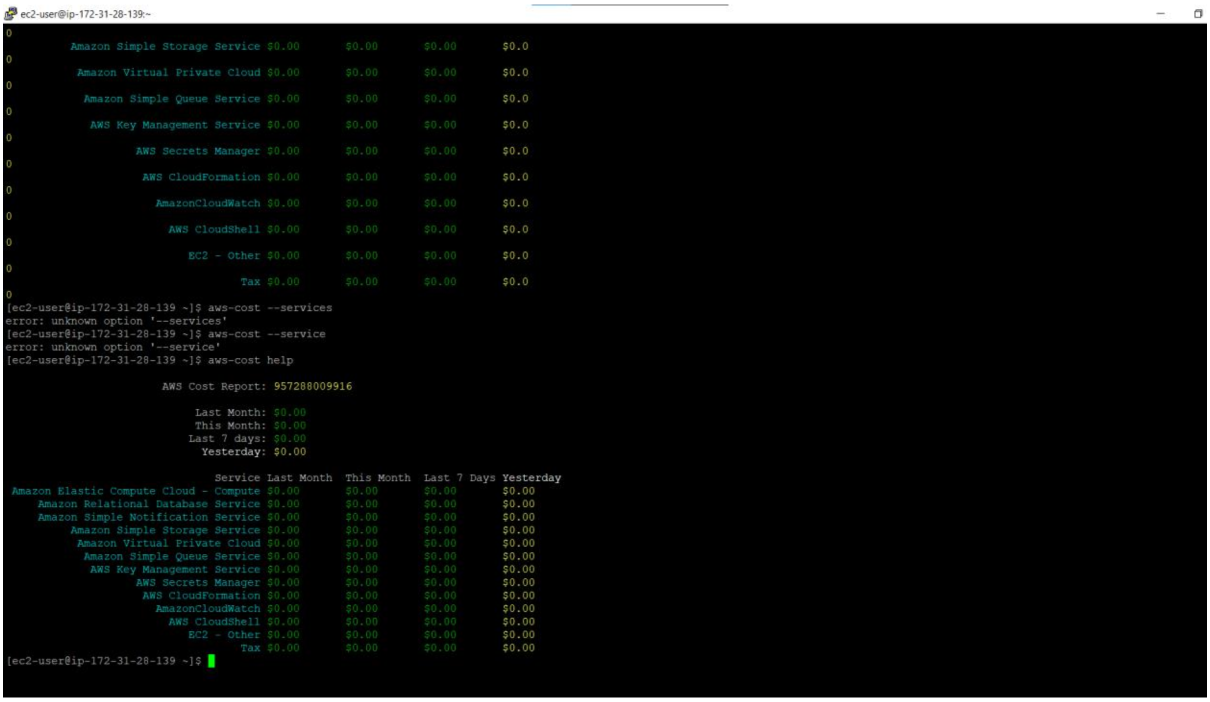
Configuration association status
The status of the State Manager associations created by your configuration.

5 Total	
Success	0
Failed	0
Pending	5

CloudWatch Dashboard



AWS Cost CLI



ec2-user@ip-172-31-28-139:~

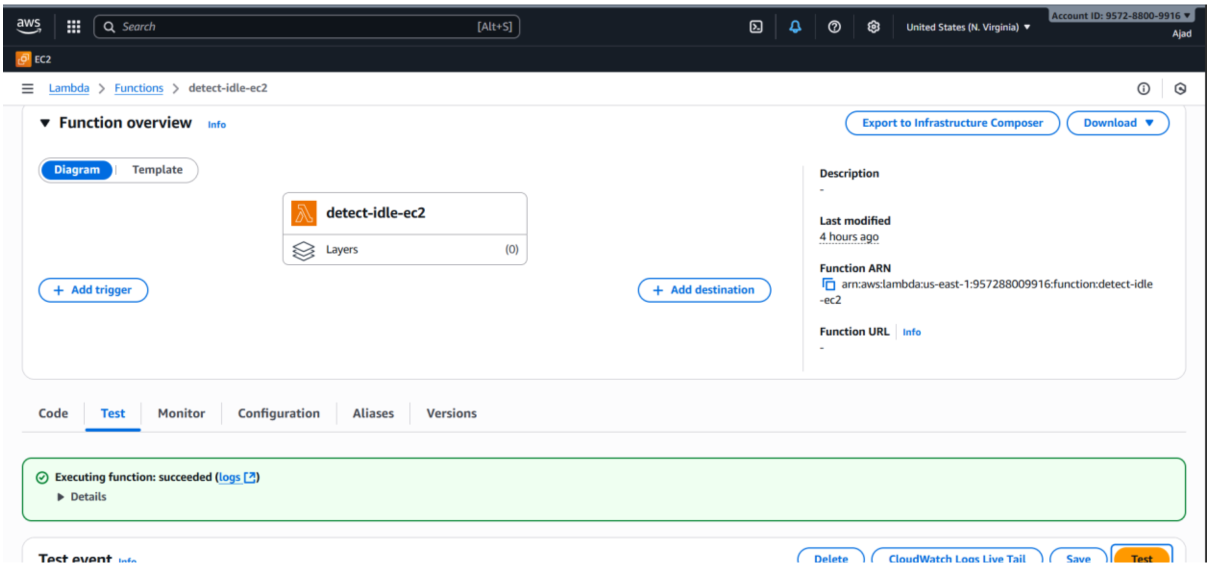
```
0
          Amazon Simple Storage Service $0.00       $0.00       $0.00       $0.0
0
           Amazon Virtual Private Cloud $0.00       $0.00       $0.00       $0.0
0
            Amazon Simple Queue Service $0.00       $0.00       $0.00       $0.0
0
             AWS Key Management Service $0.00       $0.00       $0.00       $0.0
0
                    AWS Secrets Manager $0.00       $0.00       $0.00       $0.0
0
                     AWS CloudFormation $0.00       $0.00       $0.00       $0.0
0
                       AmazonCloudWatch $0.00       $0.00       $0.00       $0.0
0
                         AWS CloudShell $0.00       $0.00       $0.00       $0.0
0
                            EC2 - Other $0.00       $0.00       $0.00       $0.0
0
                                    Tax $0.00       $0.00       $0.00       $0.0
0
[ec2-user@ip-172-31-28-139 ~]$ aws-cost --services
error: unknown option '--services'
[ec2-user@ip-172-31-28-139 ~]$ aws-cost --service
error: unknown option '--service'
[ec2-user@ip-172-31-28-139 ~]$ aws-cost help

                       AWS Cost Report: 957288009916

                            Last Month: $0.00
                            This Month: $0.00
                           Last 7 days: $0.00
                             Yesterday: $0.00

                                Service Last Month  This Month  Last 7 Days Yesterday
 Amazon Elastic Compute Cloud - Compute $0.00       $0.00       $0.00       $0.00
     Amazon Relational Database Service $0.00       $0.00       $0.00       $0.00
     Amazon Simple Notification Service $0.00       $0.00       $0.00       $0.00
          Amazon Simple Storage Service $0.00       $0.00       $0.00       $0.00
           Amazon Virtual Private Cloud $0.00       $0.00       $0.00       $0.00
            Amazon Simple Queue Service $0.00       $0.00       $0.00       $0.00
             AWS Key Management Service $0.00       $0.00       $0.00       $0.00
                    AWS Secrets Manager $0.00       $0.00       $0.00       $0.00
                     AWS CloudFormation $0.00       $0.00       $0.00       $0.00
                       AmazonCloudWatch $0.00       $0.00       $0.00       $0.00
                         AWS CloudShell $0.00       $0.00       $0.00       $0.00
                            EC2 - Other $0.00       $0.00       $0.00       $0.00
                                    Tax $0.00       $0.00       $0.00       $0.00
[ec2-user@ip-172-31-28-139 ~]$ 
```

AWS Lambda Function



detect-idle-ec2

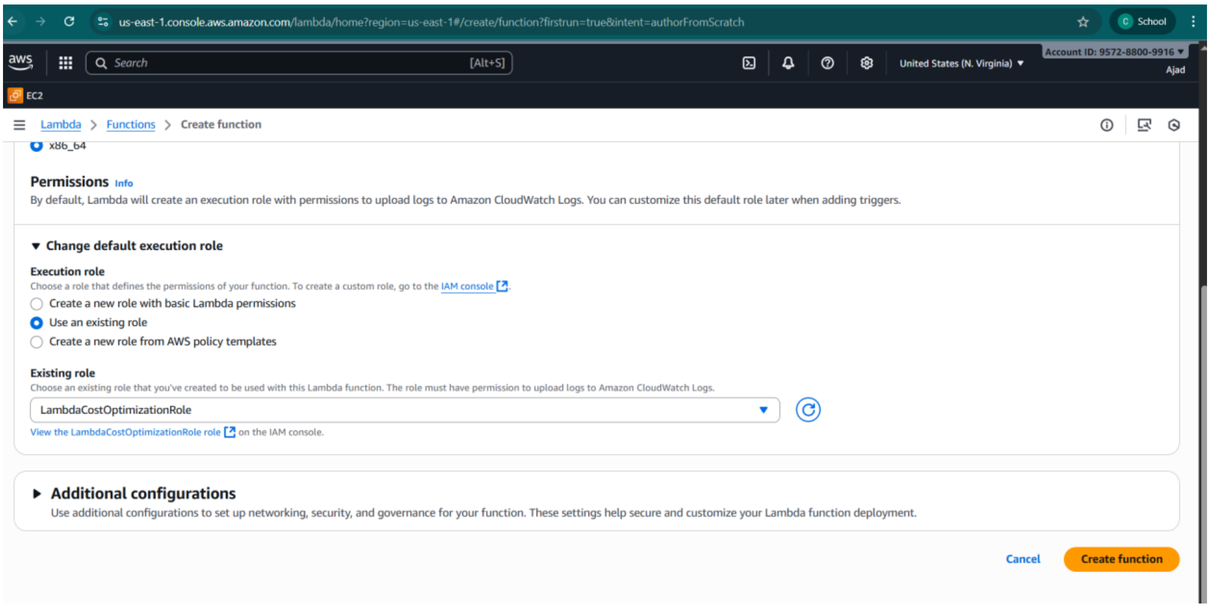
Description
-

Last modified
4 hours ago

Function ARN
arn:aws:lambda:us-east-1:957288009916:function:detect-idle-ec2

Function URL
-

Executing function: succeeded (logs)



us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function?firstrun=true&intent=authorFromScratch
School
aws
Search
[Alt+S]
United States (N. Virginia) ▾
Account ID: 9572-8800-9916 ▾
Ajad
EC2
Lambda > Functions > Create function

● x86_64

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions
● Use an existing role
○ Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

LambdaCostOptimizationRole ▾

View the LambdaCostOptimizationRole role on the IAM console.

▶ **Additional configurations**

Use additional configurations to set up networking, security, and governance for your function. These settings help secure and customize your Lambda function deployment.

CHALLENGES & SOLUTIONS

- IAM precision
  - The challenge: Write policies that are truly least-privilege yet still let Lambda and EC2 do their jobs.
  - What worked: Started from the smallest possible permission set, validated behavior with policy simulation and safe dry-runs, then added only what was proven necessary.
- Data parsing
  - The challenge: The Cost Explorer response (get_cost_and_usage) is deeply nested and hard to analyze directly.
  - What worked: Built a stepwise parser using Pandas' json_normalize to flatten the payload into a tidy DataFrame ready for charts, QA, and archival.
- Load generation
  - The challenge: Produce predictable, repeatable load on a modest t2.micro without causing instability.

  - What worked: Standardized on stress-ng with targeted flags (for example, --cpu 1 --cpu-load 50), iterating to hit specific utilization targets while keeping the instance responsive.
- Cross-service access
  - The challenge: Ensure Lambda can safely interact with EC2, CloudWatch, and Cost Explorer without over-permissioning.
  - What worked: Defined a dedicated role (LambdaCostOptimizationRole) with trusted relationships and narrowly scoped custom policies, reviewed and refined as services were integrated.

6. CONCLUSION & FUTURE ROADMAP

Conclusion

The project is on track and has moved decisively from design to execution. In nine weeks, it delivered a secure AWS foundation, a reliable monitoring pipeline, repeatable load generation, a cost data workflow, and a working visualization layer. These pieces now form a cohesive platform on which the core contribution—intelligent, approval-aware automation—will be built next.

What this means in practice:

- The environment is safe by default (least-privilege IAM, controlled automation).
- The signals needed for decisions (metrics, costs, tags) are flowing and validated.
- The dashboard provides immediate visibility and a path to action.

Next, the focus shifts to decision logic and automation quality:

- Codify idle and stale-resource rules with tunable thresholds.
- Add approval workflows and guardrails to make cleanup safe and auditable.
- Quantify savings with before/after comparisons surfaced directly in the dashboard.

Plan for Weeks 10-16

Week	Task	Deliverable
10-11	Develop Lambda functions for idle instance detection and stale resource scanning.	Python scripts using Boto3 to identify idle EC2s and stale EIPs, EBS, Snapshots.
12	Develop the cleanup Lambda function and integrate with DynamoDB for logging.	A secure function to stop/delete resources, triggered manually based on DynamoDB records.
13	Implement SNS notifications for alerting and user approval workflows.	Email alerts sent to users when resources are flagged for cleanup.
14	Migrate data storage from local files to Amazon S3 and DynamoDB.	Cost data stored in S3, cleanup candidates logged in DynamoDB.
15	Enhance Streamlit dashboard with interactive controls and historical analysis.	Dashboard with buttons to trigger cleanup and view savings reports.
16	End-to-end testing, cost-benefit analysis, final documentation, and report submission.	Final system test report, dissertation document, and presentation.

REFERENCES

1. AWS Well-Architected Framework (2023). *Pillar: Cost Optimization*. Amazon Web Services.

2. Sbarski, P. (2017). *Serverless Architectures on AWS*. Manning Publications.

3. Armbrust, M., et al. (2010). *A view of cloud computing*. Communications of the ACM.

4. Li, Z., et al. (2020). *Serverless Resilience: Using AWS Lambda for Event-Driven Recovery*. IEEE Cloud Computing.

5. Gartner (2022). *Avoiding Common Pitfalls in Cloud IaaS Migration*.

6. Boto3 Documentation (2025). *AWS SDK for Python*. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html