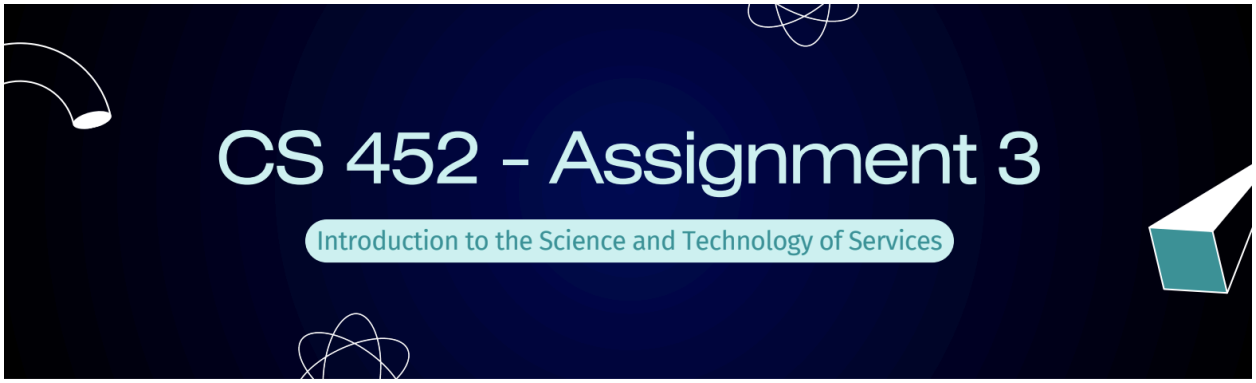



**Winter Semester 2025-2026**

**Name:** Panagiotis Kalogiannis

**ID:** csd4888

**Date:** January 17, 2026

# Cloud Resource Allocation Project Report

## 1. Introduction

This report presents the implementation and results of a game-theoretic resource allocation system for cloud computing services. The project implements a two-step optimization process where three users (tasks) compete for five computational resources provided by a cloud provider.

## 2. Configuration

## 2.1 Student ID and Weight Calculation

**Student ID Last Two Digits (D)**: 88

**Weight Calculations**:

| Task | Weight Time (wt) | Weight Expense (we) |
|---|---|---|
| S1 | D/100 = 0.88 | 1 - wt = 0.12 |
| S2 | (D+1)%100/100 = 0.89 | 1 - wt = 0.11 |
| S3 | (D+2)%100/100 = 0.90 | 1 - wt = 0.10 |

## 2.2 Task Configuration

| Task | Subtasks | Deadline (s) | Budget (€) |
|---|---|---|---|
| S1 | 2 | 500 | 20 |
| S2 | 3 | 300 | 30 |
| S3 | 4 | 800 | 30 |

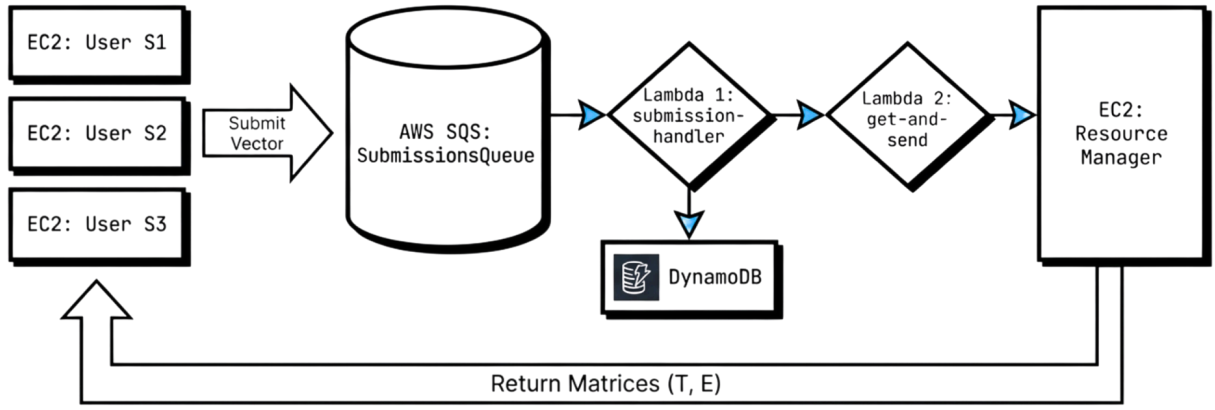
## 2.3 Resource Configuration

| Resource | Price (€/s) | Execution Times (S1, S2, S3) |
|---|---|---|
| R1 | 1.0 | 5.0, 6.0, 4.0 |
| R2 | 1.2 | 4.2, 5.0, 3.5 |
| R3 | 1.5 | 3.6, 4.0, 3.2 |
| R4 | 1.8 | 3.0, 3.5, 2.8 |
| R5 | 2.0 | 2.8, 3.0, 2.4 |

---

# 3. Architecture

## 3.1 System Architecture Diagram




Workflow: Asynchronous Event-Driven Architecture  Users optimize locally -> Submit to Queue -> Aggregated in DynamoDB -> Batch Processed by Manager -> Results Distributed.

**Architecture Flow Description**:

1. **Submission Phase**: Users (S1, S2, S3) perform local optimization and submit allocation vectors to SQS queue
2. **Processing Phase**: Lambda 1 (submission-handler) is triggered by SQS messages, stores allocations in DynamoDB

3. **Aggregation Phase**: When all 3 users have submitted, Lambda 1 triggers Lambda 2 (get-and-send)
4. **Calculation Phase**: Lambda 2 retrieves allocations from DynamoDB and sends them to Resource Manager
5. **Distribution Phase**: Resource Manager calculates actual execution times and expenses (accounting for multiplexing) and sends T and E matrices back to all users
6. **Status Update**: Lambda 2 updates allocation status in DynamoDB to 'processed'

The system consists of:

1. **User Layer**: 3 EC2 instances running Flask applications (one per user)
2. **Message Queue**: SQS queue for asynchronous communication
3. **Processing Layer**: 2 Lambda functions for workflow orchestration
4. **Storage Layer**: DynamoDB for persistent allocation storage
5. **Provider Layer**: Resource Manager EC2 instance for matrix calculations

# 3.2 Component Description

**User EC2 Instances**:

- Perform brute-force optimization to find best allocation
- Submit allocation vectors to SQS
- Receive and process T and E matrices from provider
- Calculate actual utilities

**Lambda 1 (Submission Handler)**:

- Triggered by SQS messages
- Stores allocations in DynamoDB
- Checks if all users submitted
- Triggers Lambda 2 when ready

**Lambda 2 (Get and Send)**:

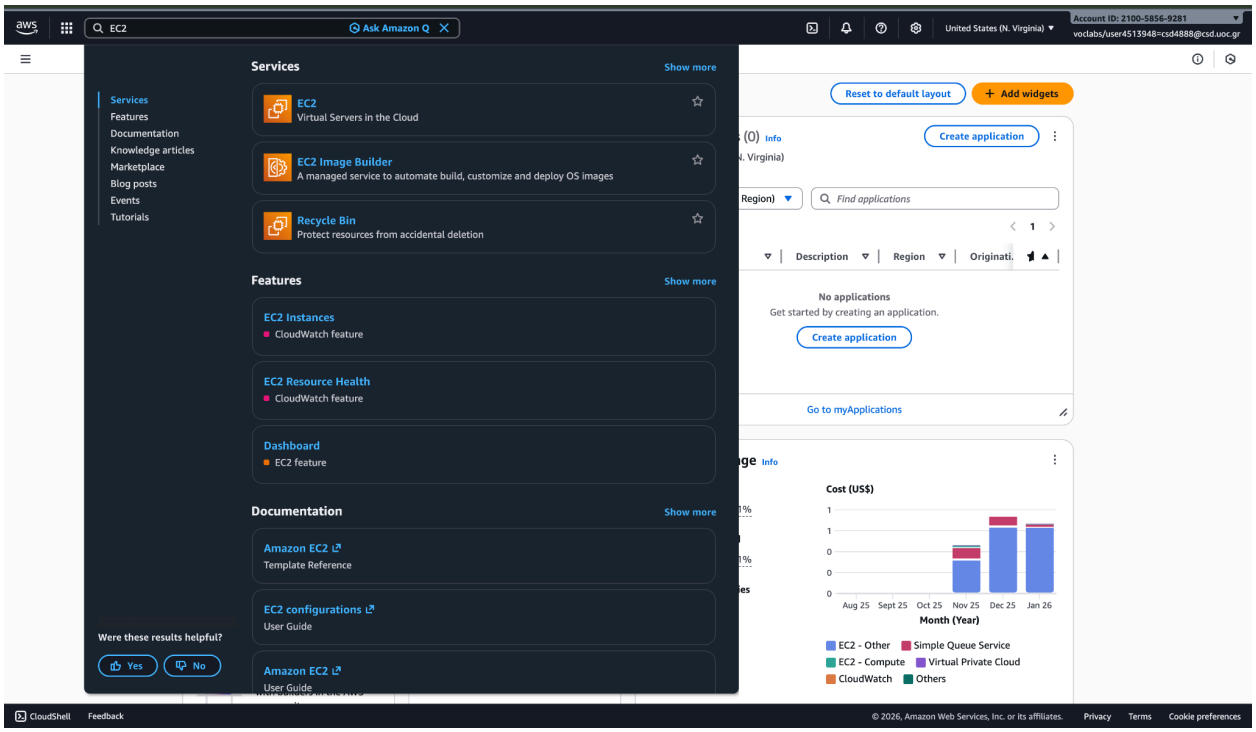
- Retrieves allocations from DynamoDB
- Sends to Resource Manager
- Updates processing status

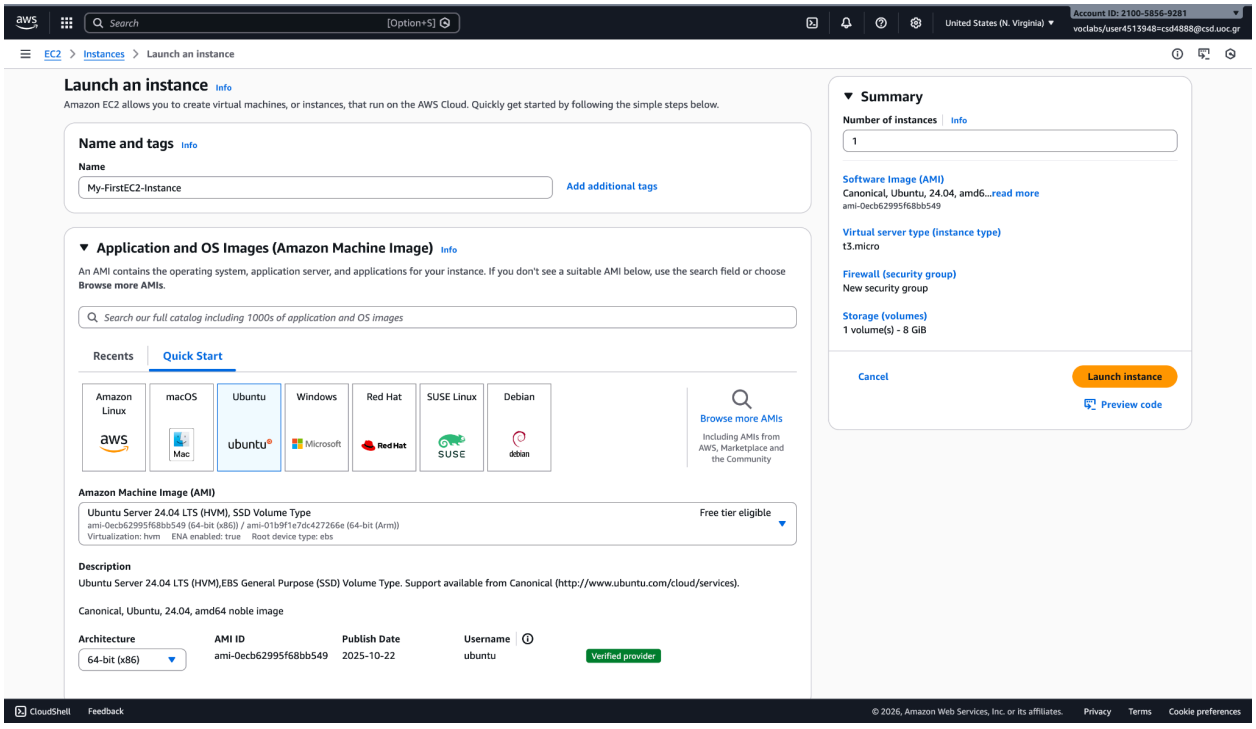
**Resource Manager EC2**:

- Receives allocation vectors from all users
- Calculates actual execution times (with multiplexing)
- Calculates expenses
- Returns T and E matrices to users

# 4. Implementation

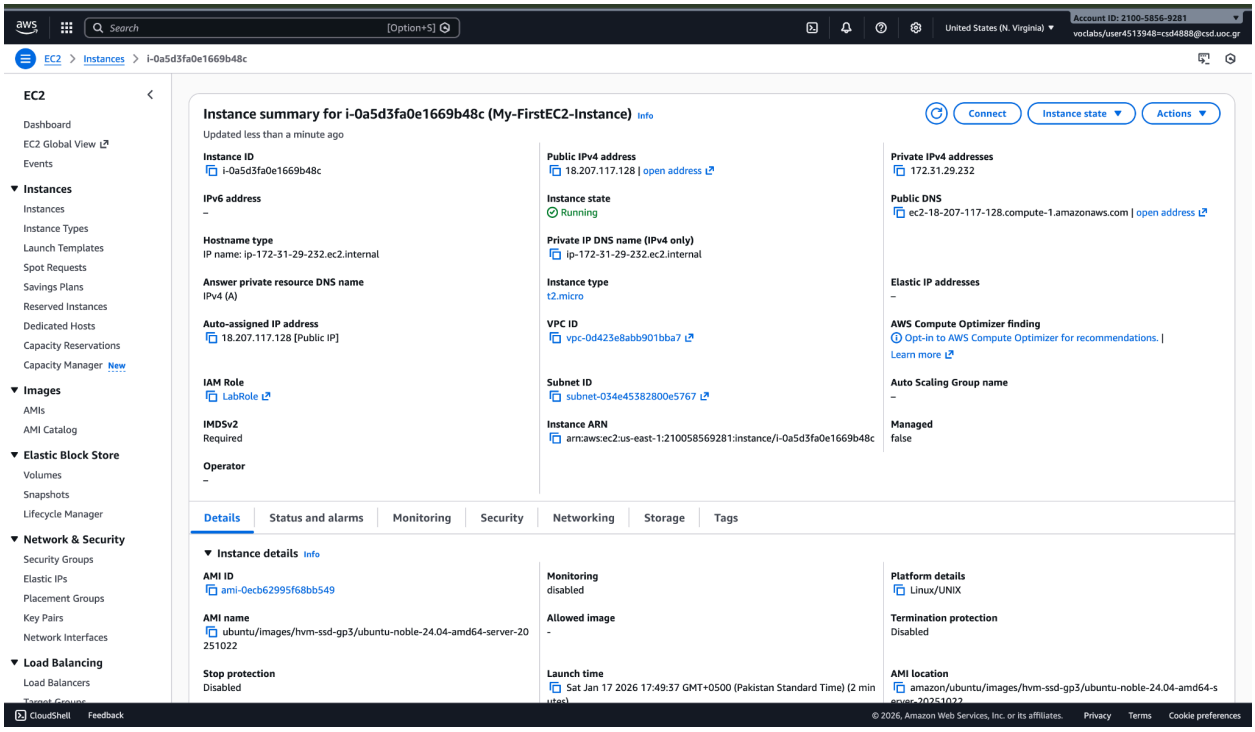
## 4.1 AWS Resources Created

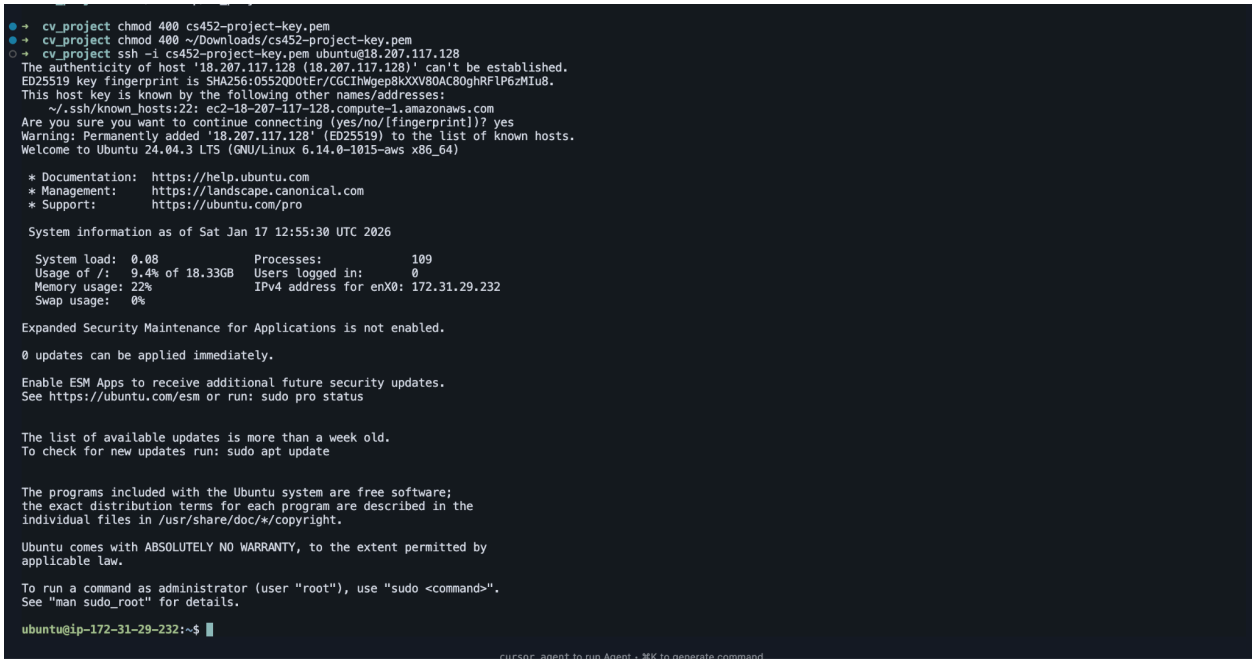


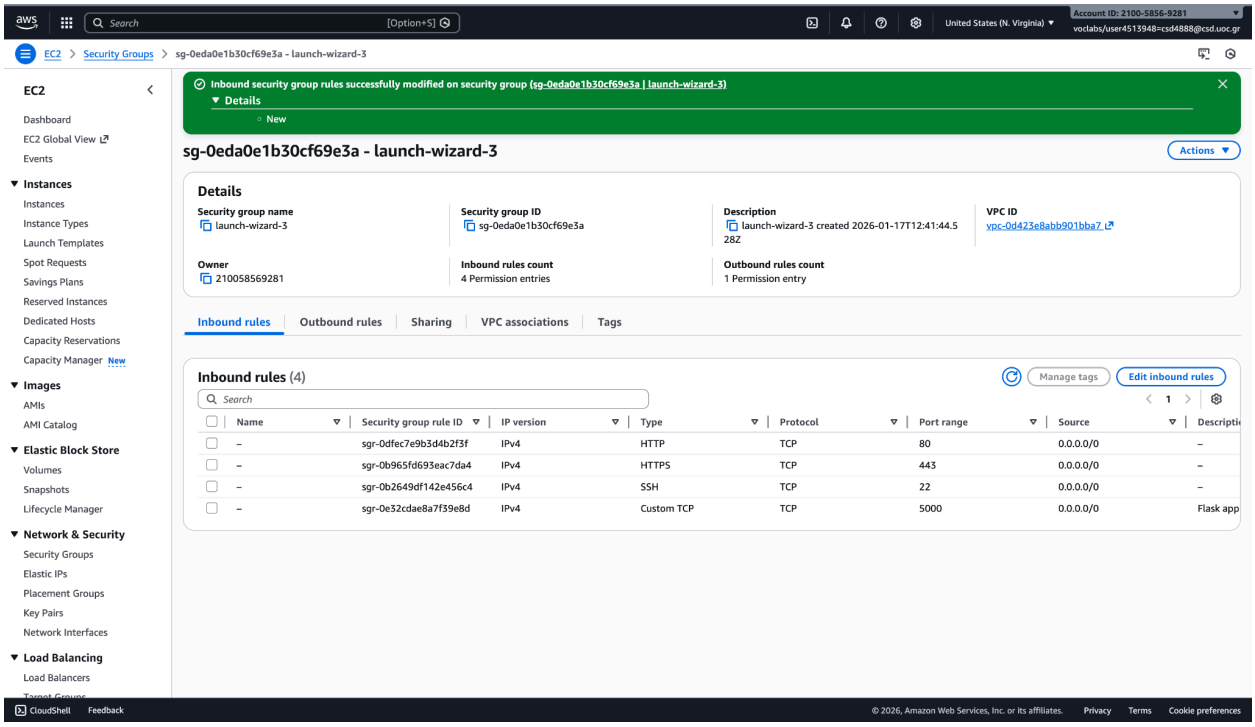



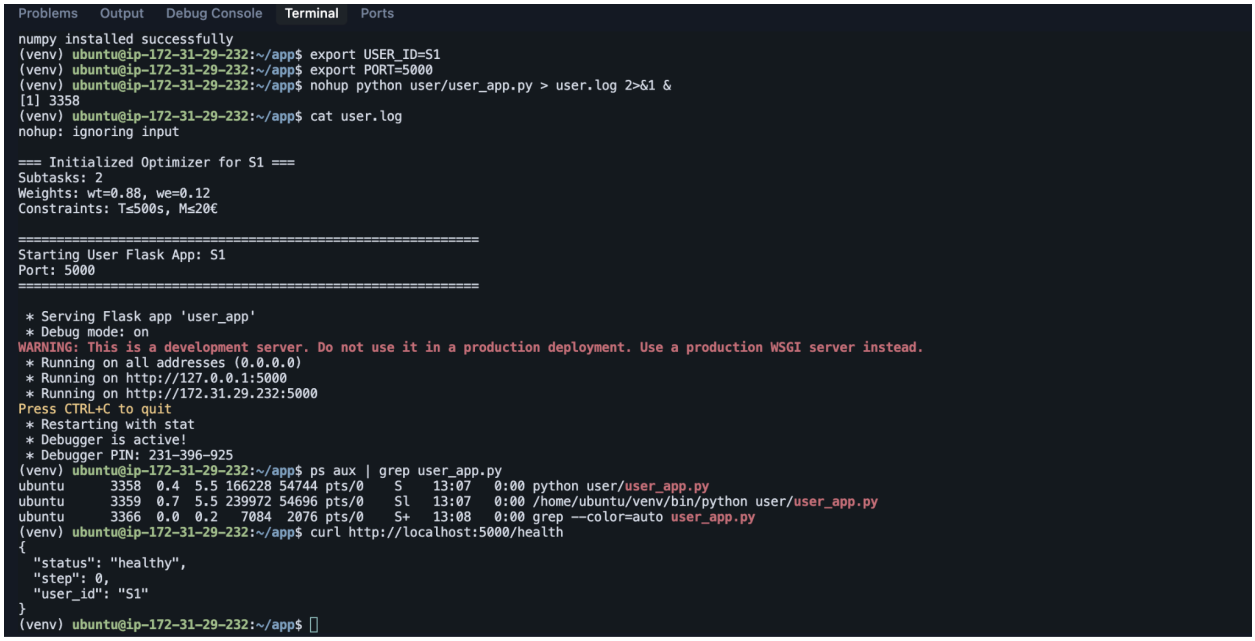
**EC2 Instances**:

- User-S1: 18.207.117.128 (ec2-18-207-117-128.compute-1.amazonaws.com)

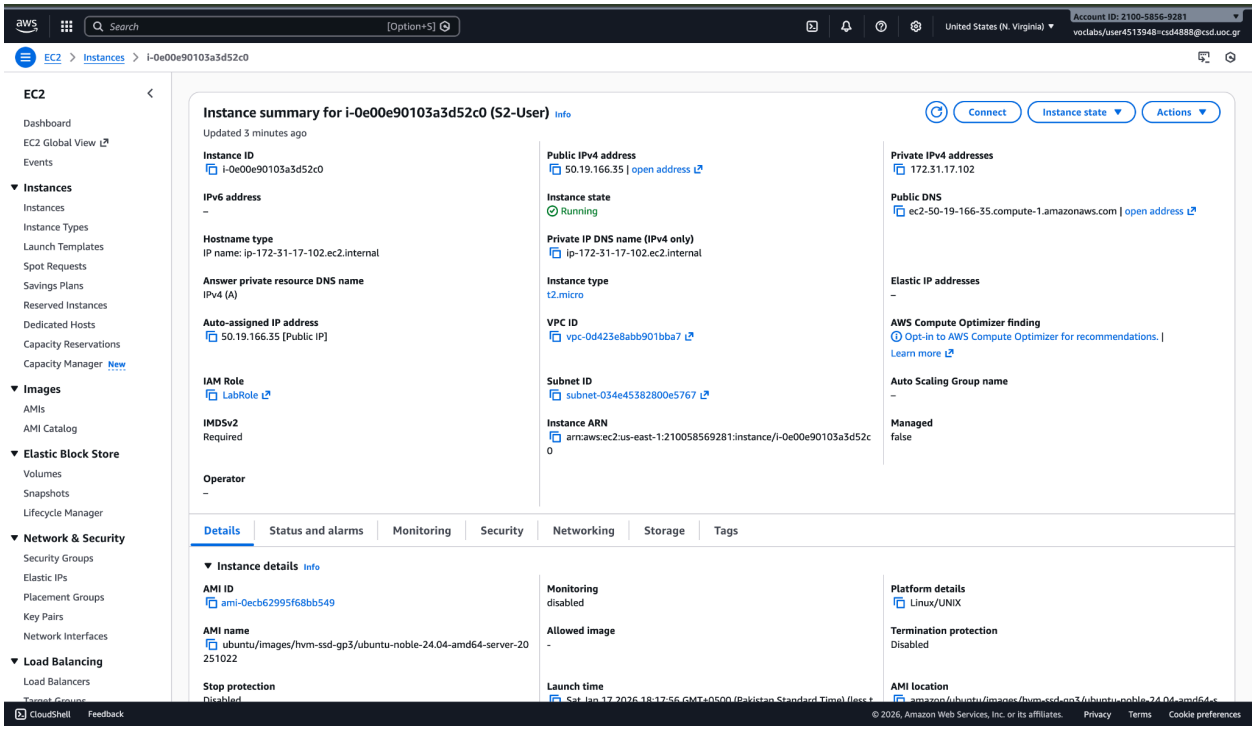


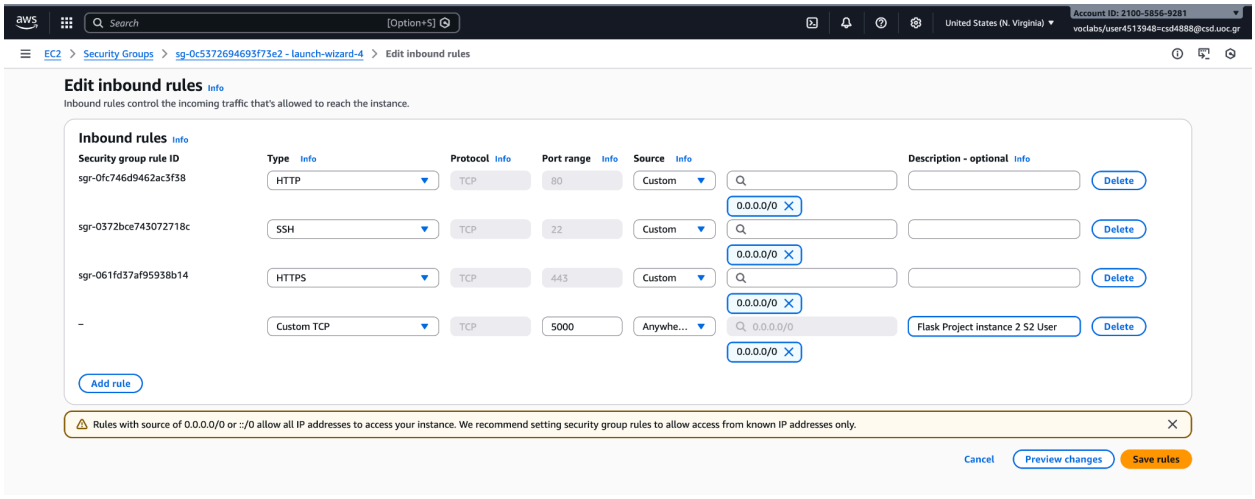





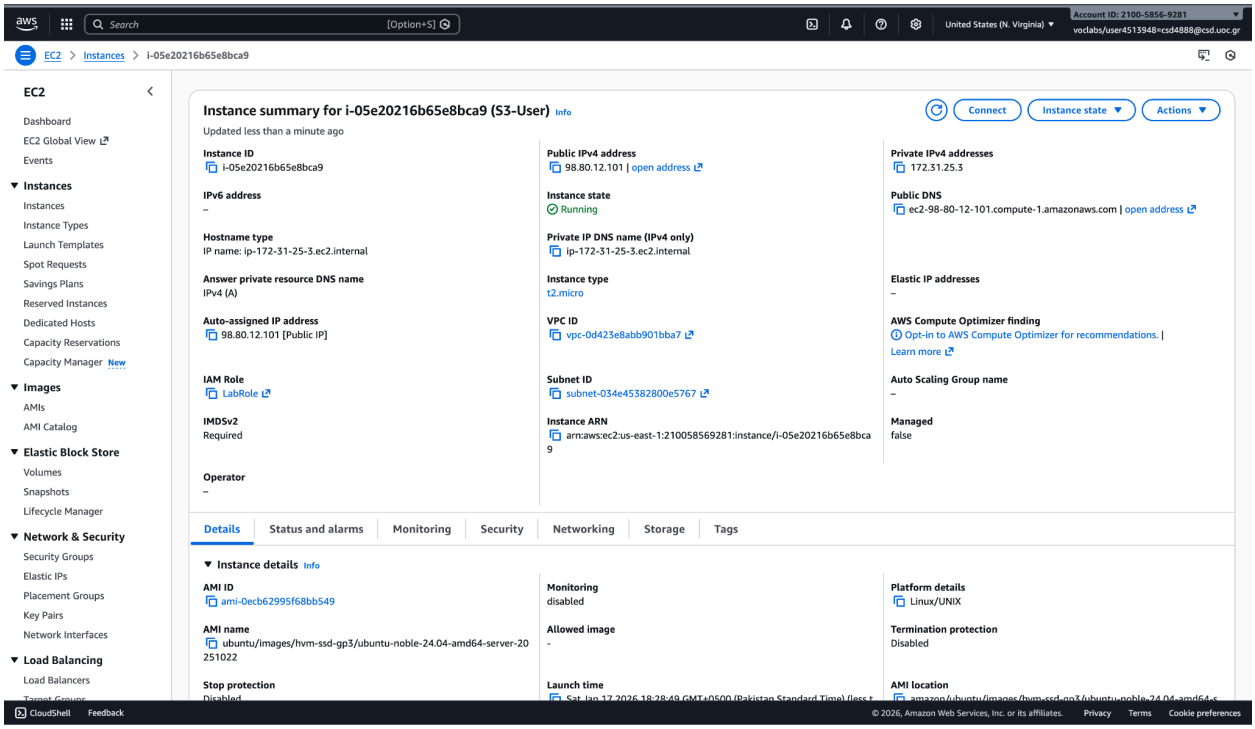

- User-S2: 50.19.166.35 (ec2-50-19-166-35.compute-1.amazonaws.com)



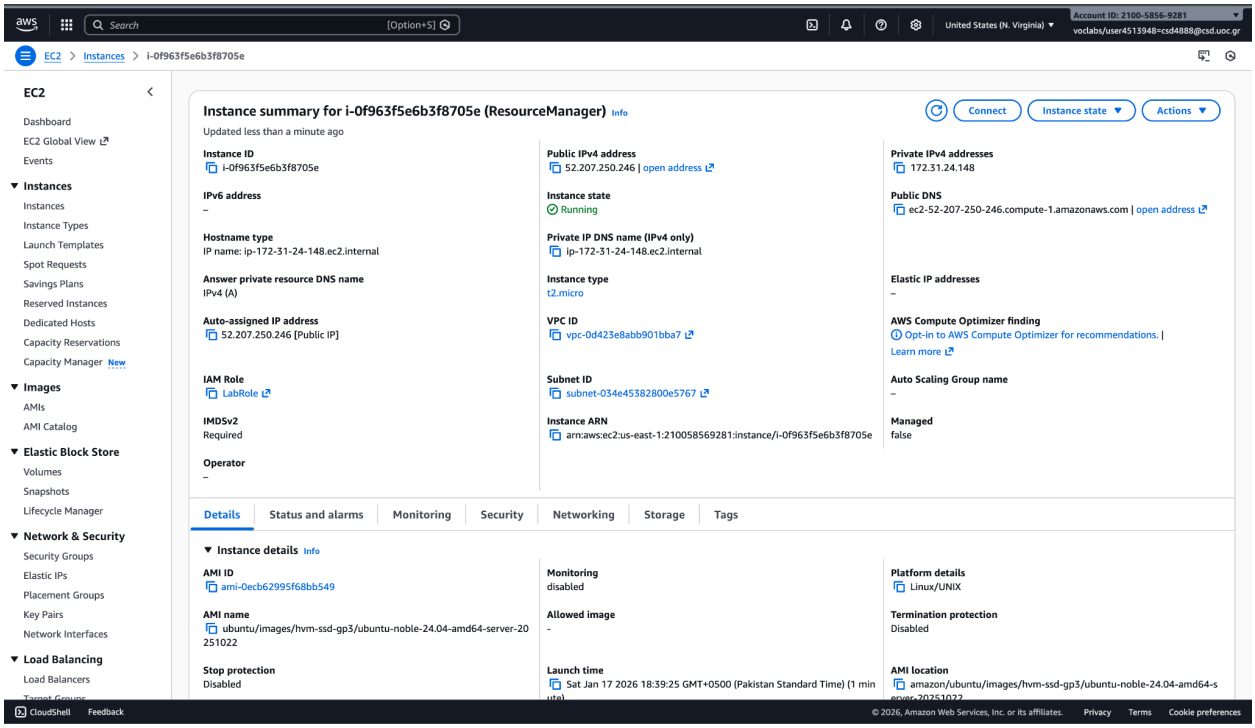

```
Successfully installed Flask-3.1.2 blinker-1.9.0 boto3-1.42.30 botocore-1.42.30 certifi-2026.1.4 charset_normalizer-3.4.4 click-8.3.1 idna-3.11 itsdangerous-2.2.0 jinja2-3.1.6 jme
spath-1.0.1 markupsafe-3.0.3 numpy-2.4.1 python-dateutil-2.9.0.post0 requests-2.32.5 s3transfer-0.16.0 six-1.17.0 urllib3-2.6.3 werkzeug-3.1.5
(venv) ubuntu@ip-172-31-17-102:~$ mkdir -p ~/app
(venv) ubuntu@ip-172-31-17-102:~$ cd ~/app
(venv) ubuntu@ip-172-31-17-102:~/app$ export USER_ID=S2
(venv) ubuntu@ip-172-31-17-102:~/app$ export PORT=5000
(venv) ubuntu@ip-172-31-17-102:~/app$ nohup python user/user_app.py > user.log 2>&1 &
[1] 3191
(venv) ubuntu@ip-172-31-17-102:~/app$ cat user.log
nohup: ignoring input

=== Initialized Optimizer for S2 ===
Subtasks: 3
Weights: wt=0.89, we=0.11
Constraints: T≤300s, M≤30€

============================================================
Starting User Flask App: S2
Port: 5000
============================================================

 * Serving Flask app 'user_app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.17.102:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 865-556-343
(venv) ubuntu@ip-172-31-17-102:~/app$
```

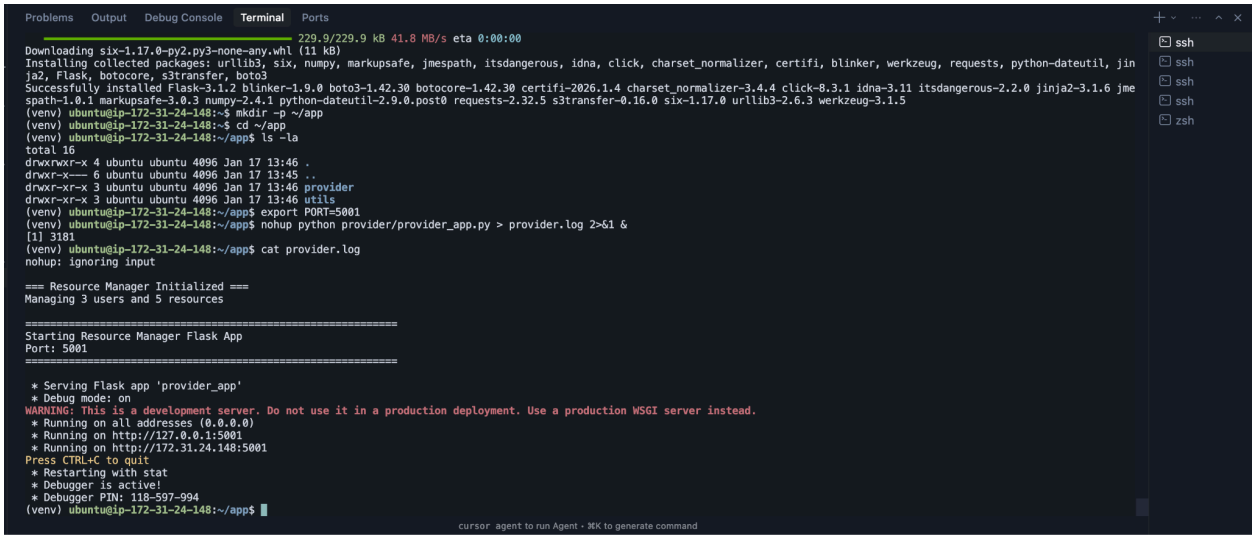
- User-S3: 98.80.12.101 (ec2-98-80-12-101.compute-1.amazonaws.com)



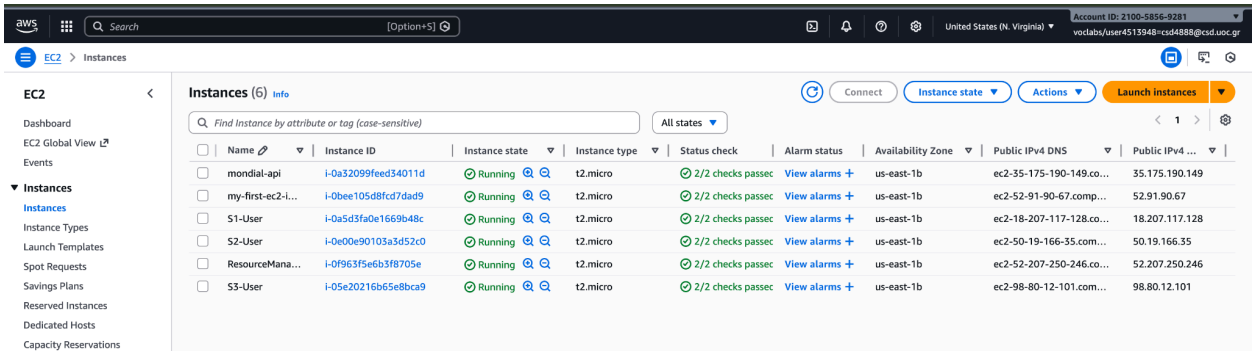


- ResourceManager: 52.207.250.246 (ec2-52-207-250-246.compute-1.amazonaws.com)


Search
[Option+S]
United States (N. Virginia)
Account ID: 2100-5856-9281
voclabs/user4513948=csd4888@csd.uoc.gr
EC2 > Instances > i-0f963f5e6b3f8705e
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Capacity Manager New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Instance summary for i-0f963f5e6b3f8705e (ResourceManager) Info
Updated less than a minute ago
Connect
Instance state
Actions
Instance ID
i-0f963f5e6b3f8705e
IPv6 address
–
Hostname type
IP name: ip-172-31-24-148.ec2.internal
Answer private resource DNS name
IPv4 (A)
Auto-assigned IP address
52.207.250.246 [Public IP]
IAM Role
LabRole
IMDSv2
Required
Operator
–
Public IPv4 address
52.207.250.246 | open address
Instance state
Running
Private IP DNS name (IPv4 only)
ip-172-31-24-148.ec2.internal
Instance type
t2.micro
VPC ID
vpc-0d423e8abb901bba7
Subnet ID
subnet-034e45382800e5767
Instance ARN
arn:aws:ec2:us-east-1:210058569281:instance/i-0f963f5e6b3f8705e
Private IPv4 addresses
172.31.24.148
Public DNS
ec2-52-207-250-246.compute-1.amazonaws.com | open address
Elastic IP addresses
–
AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more
Auto Scaling Group name
–
Managed
false
Details
Status and alarms
Monitoring
Security
Networking
Storage
Tags
Instance details Info
AMI ID
ami-0ecb62995f68bb549
AMI name
ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20251022
Stop protection
Disabled
Monitoring
disabled
Allowed image
-
Launch time
Sat Jan 17 2026 18:39:25 GMT+0500 (Pakistan Standard Time) (1 minute)
Platform details
Linux/UNIX
Termination protection
Disabled
AMI location
amazon/ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20251022
CloudShell
Feedback
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

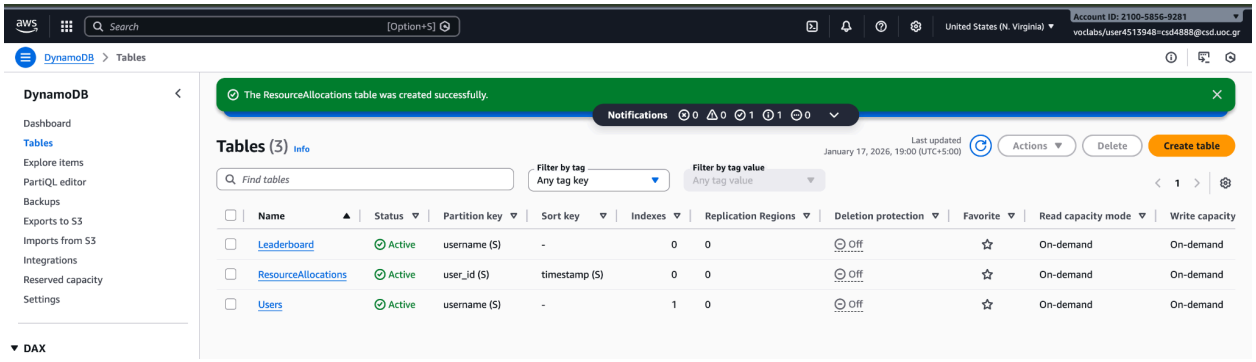

Problems
Output
Debug Console
Terminal
Ports
229.9/229.9 kB 41.8 MB/s eta 0:00:00
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: urllib3, six, numpy, markupsafe, jmespath, itsdangerous, idna, click, charset_normalizer, certifi, blinker, werkzeug, requests, python-dateutil, jinja2, Flask, botocore, s3transfer, boto3
Successfully installed Flask-3.1.2 blinker-1.9.0 boto3-1.42.30 botocore-1.42.30 certifi-2026.1.4 charset_normalizer-3.4.4 click-8.3.1 idna-3.11 itsdangerous-2.2.0 jinja2-3.1.6 jmespath-1.0.1 markupsafe-3.0.3 numpy-2.4.1 python-dateutil-2.9.0.post0 requests-2.32.5 s3transfer-0.16.0 six-1.17.0 urllib3-2.6.3 werkzeug-3.1.5
(venv) ubuntu@ip-172-31-24-148:~$ mkdir -p ~/app
(venv) ubuntu@ip-172-31-24-148:~$ cd ~/app
(venv) ubuntu@ip-172-31-24-148:~/app$ ls -la
total 16
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 17 13:46 .
drwxr-x--- 6 ubuntu ubuntu 4096 Jan 17 13:45 ..
drwxr-xr-x 3 ubuntu ubuntu 4096 Jan 17 13:46 provider
drwxr-xr-x 3 ubuntu ubuntu 4096 Jan 17 13:46 utils
(venv) ubuntu@ip-172-31-24-148:~/app$ export PORT=5001
(venv) ubuntu@ip-172-31-24-148:~/app$ nohup python provider/provider_app.py > provider.log 2>&1 &
[1] 3181
(venv) ubuntu@ip-172-31-24-148:~/app$ cat provider.log
nohup: ignoring input
=== Resource Manager Initialized ===
Managing 3 users and 5 resources
Starting Resource Manager Flask App
Port: 5001
* Serving Flask app 'provider_app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5001
* Running on http://172.31.24.148:5001
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 118-597-994
(venv) ubuntu@ip-172-31-24-148:~/app$
ssh
ssh
ssh
ssh
zsh
cursor agent to run Agent • ⌘K to generate command

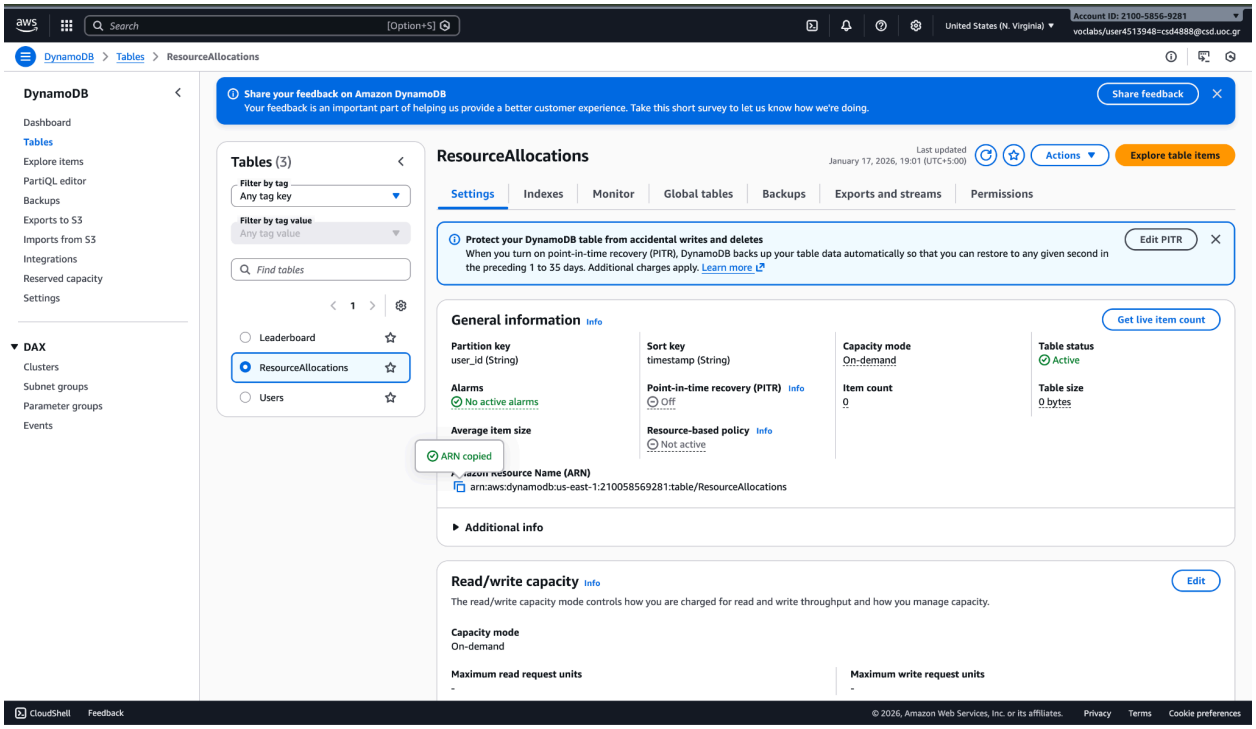
## Overall:


Search
[Option+S]
United States (N. Virginia)
Account ID: 2100-5856-9281
voclabs/user4513948=csd4888@csd.uoc.gr
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Instances (6) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
mondial-api
i-0a32099feed34011d
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-35-175-190-149.co...
35.175.190.149
my-first-ec2-i...
i-0bee105d8fcd7dad9
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-52-91-90-67.comp...
52.91.90.67
S1-User
i-0a5d3fa0e1669b48c
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-18-207-117-128.co...
18.207.117.128
S2-User
i-0e00e90103a3d52c0
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-50-19-166-35.com...
50.19.166.35
ResourceMana...
i-0f963f5e6b3f8705e
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-52-207-250-246.co...
52.207.250.246
S3-User
i-05e20216b65e8bca9
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
ec2-98-80-12-101.com...
98.80.12.101

**DynamoDB Table**: ResourceAllocations

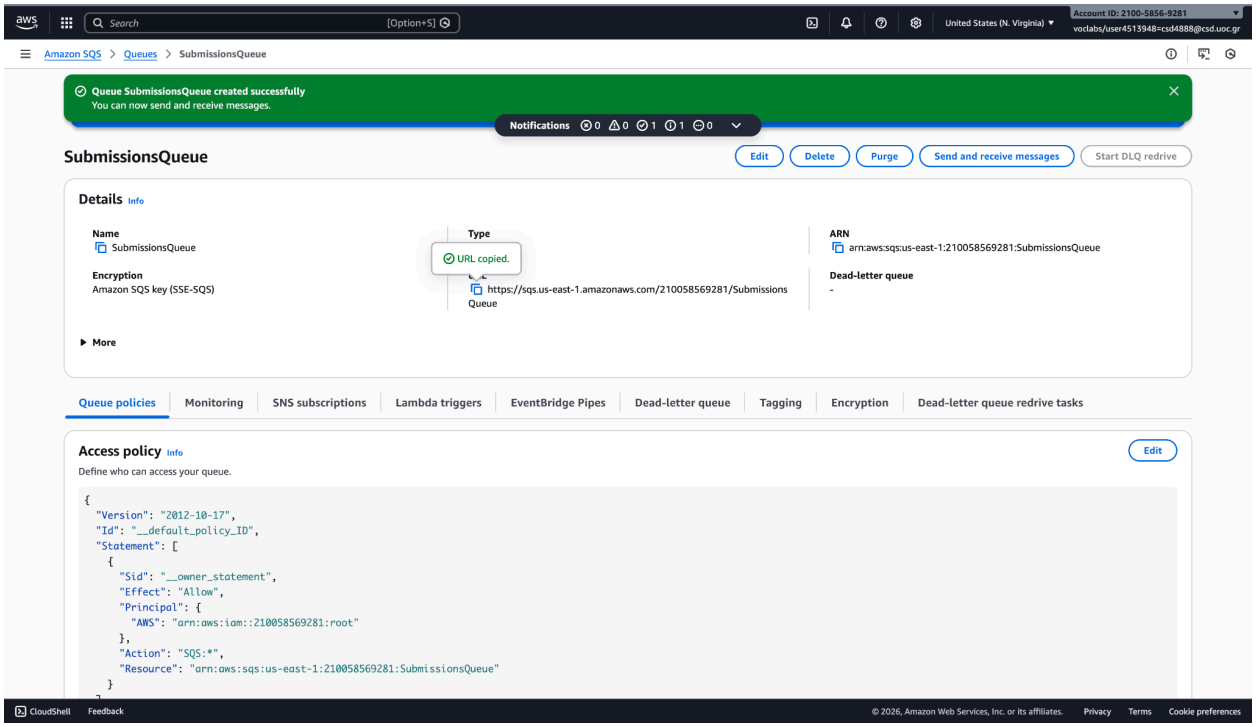
- Table Name: ResourceAllocations
- Partition Key: user_id (String)
- Sort Key: timestamp (String)
- Region: us-east-1
- Table ARN: arn:aws:dynamodb:us-east-1:210058569281:table/ResourceAllocations



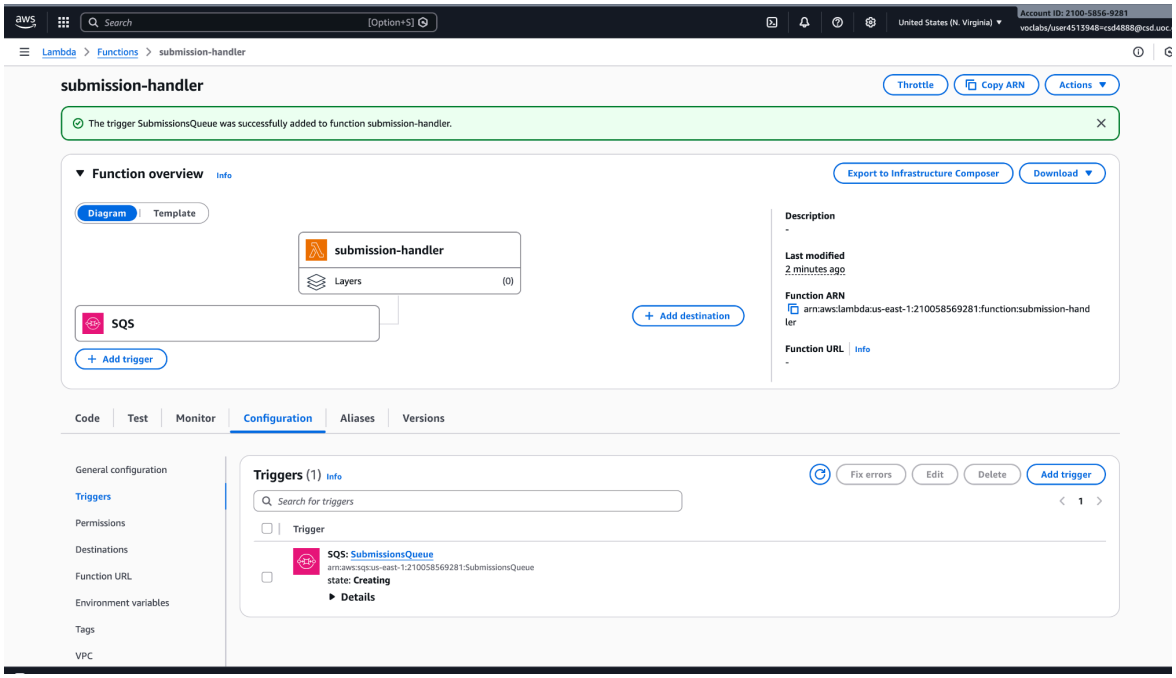
**SQS Queue**: SubmissionsQueue

- Queue Name: SubmissionsQueue
- Type: Standard Queue
- Queue URL: https://sqs.us-east-1.amazonaws.com/210058569281/SubmissionsQueue
- Region: us-east-1
- Messages processed: Multiple (handled by Lambda 1 trigger)

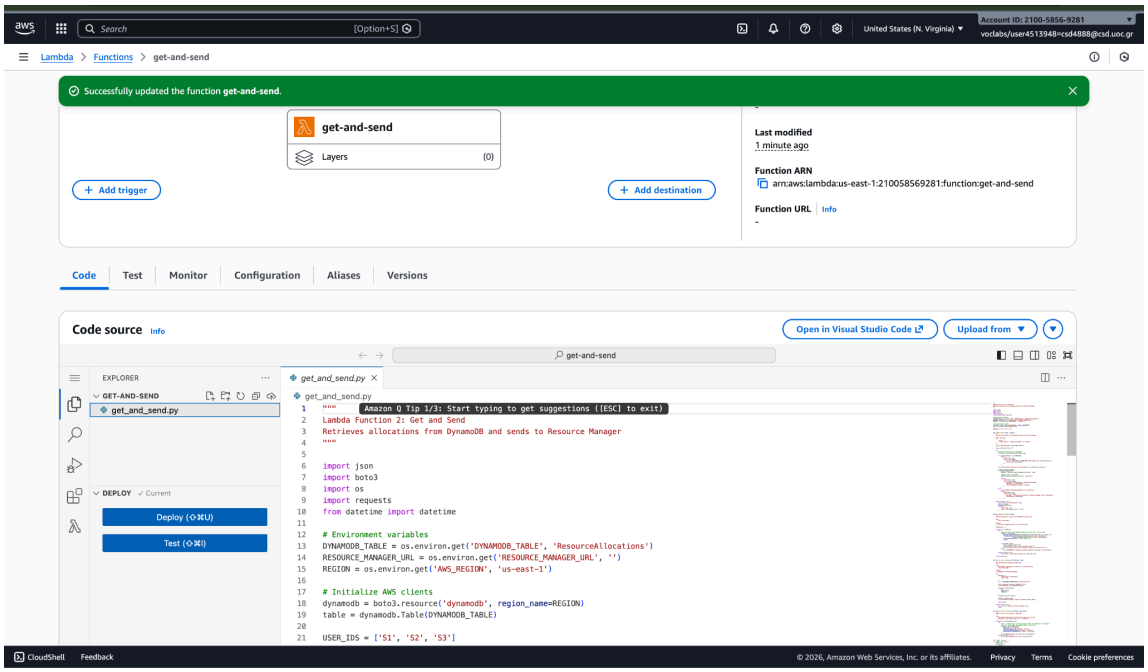



**Lambda Functions**:

- **submission-handler** (Lambda 1):
  - Function Name: submission-handler
  - Runtime: Python 3.11
  - Trigger: SQS Queue (SubmissionsQueue)
  - Environment Variables: DYNAMODB_TABLE, LAMBDA_FUNCTION_2, AWS_REGION
  - Role: Uses LabRole with DynamoDB, SQS, and Lambda permissions




- **get-and-send** (Lambda 2):
  - Function Name: get-and-send
  - Runtime: Python 3.11
  - Trigger: Invoked by Lambda 1
  - Environment Variables: DYNAMODB_TABLE, RESOURCE_MANAGER_URL, AWS_REGION
  - Timeout: 30 seconds



## 4.2 Key Code Implementation

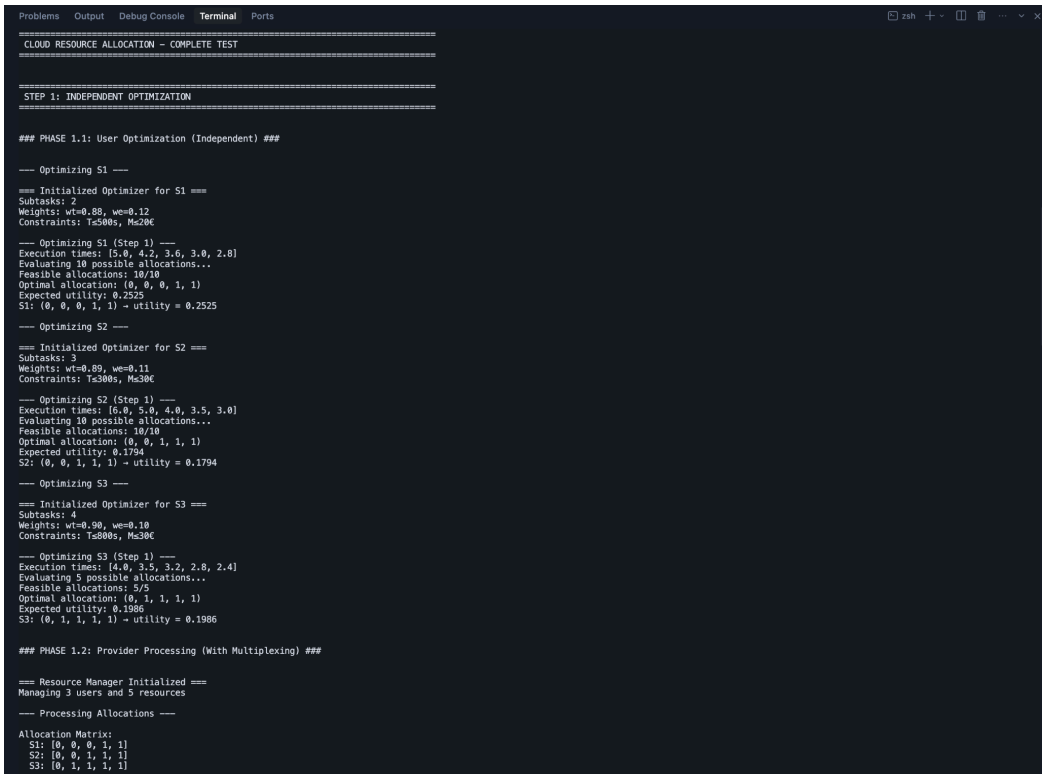
**Utility Calculation Function**:

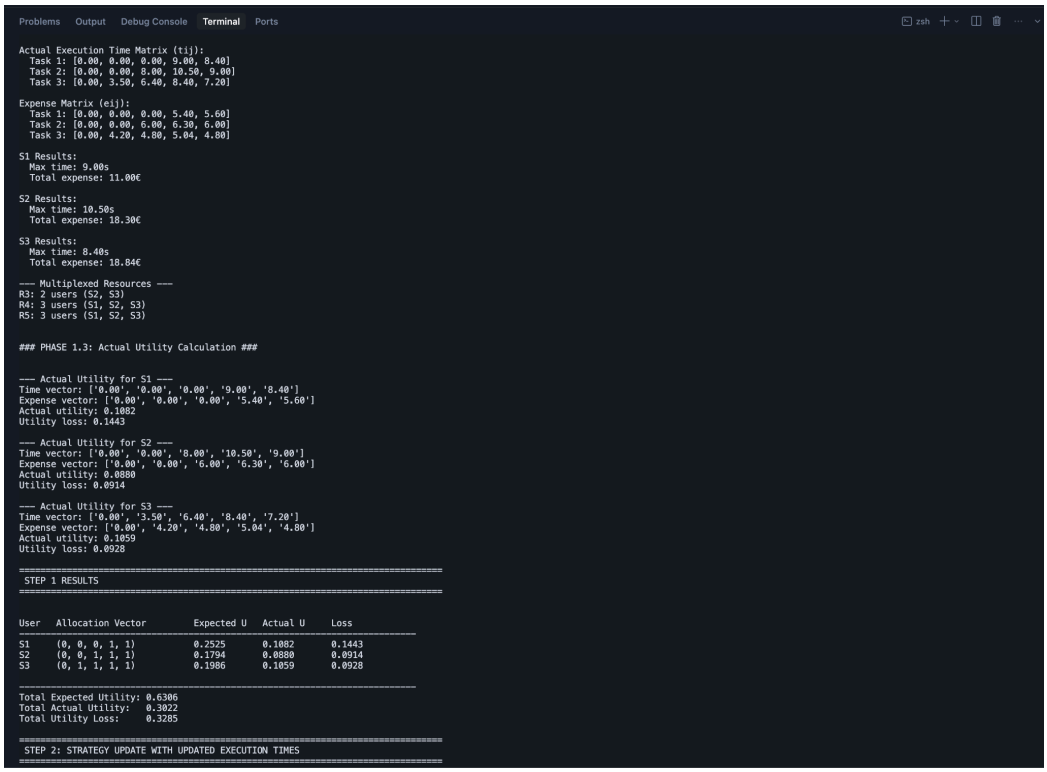
```
def calculate_utility(allocation_vector, execution_times, resource_prices,
                      weight_time, weight_expense):
    time_vector = calculate_execution_time_vector(allocation_vector,
execution_times)
    expense_vector = calculate_expense_vector(allocation_vector,
execution_times, resource_prices)

    max_time = max(time_vector)
    total_expense = sum(expense_vector)

    denominator = weight_time * max_time + weight_expense * total_expense
    return 1.0 / denominator if denominator > 0 else 0
```

**Multiplexing Calculation**:

```
def calculate_actual_execution_matrix(allocation_matrix,
base_execution_times):
    for i in range(n_tasks):
        for j in range(m_resources):
            multiplexing_factor = sum(allocation_matrix[k][j] for k in
range(n_tasks))
            if allocation_matrix[i][j] == 1:
                actual_time = multiplexing_factor *
base_execution_times[i][j]
```

# 5. Results


Problems Output Debug Console Terminal Ports
zsh
CLOUD RESOURCE ALLOCATION - COMPLETE TEST
STEP 1: INDEPENDENT OPTIMIZATION
### PHASE 1.1: User Optimization (Independent) ###
--- Optimizing S1 ---
=== Initialized Optimizer for S1 ===
Subtasks: 2
Weights: wt=0.88, we=0.12
Constraints: T≤500s, M≤20€
--- Optimizing S1 (Step 1) ---
Execution times: [5.0, 4.2, 3.6, 3.0, 2.8]
Evaluating 10 possible allocations...
Feasible allocations: 10/10
Optimal allocation: (0, 0, 0, 1, 1)
Expected utility: 0.2525
S1: (0, 0, 0, 1, 1) → utility = 0.2525
--- Optimizing S2 ---
=== Initialized Optimizer for S2 ===
Subtasks: 3
Weights: wt=0.89, we=0.11
Constraints: T≤300s, M≤30€
--- Optimizing S2 (Step 1) ---
Execution times: [6.0, 5.0, 4.0, 3.5, 3.0]
Evaluating 10 possible allocations...
Feasible allocations: 10/10
Optimal allocation: (0, 0, 1, 1, 1)
Expected utility: 0.1794
S2: (0, 0, 1, 1, 1) → utility = 0.1794
--- Optimizing S3 ---
=== Initialized Optimizer for S3 ===
Subtasks: 4
Weights: wt=0.90, we=0.10
Constraints: T≤800s, M≤30€
--- Optimizing S3 (Step 1) ---
Execution times: [4.0, 3.5, 3.2, 2.8, 2.4]
Evaluating 5 possible allocations...
Feasible allocations: 5/5
Optimal allocation: (0, 1, 1, 1, 1)
Expected utility: 0.1986
S3: (0, 1, 1, 1, 1) → utility = 0.1986
### PHASE 1.2: Provider Processing (With Multiplexing) ###
=== Resource Manager Initialized ===
Managing 3 users and 5 resources
--- Processing Allocations ---
Allocation Matrix:
S1: [0, 0, 0, 1, 1]
S2: [0, 0, 1, 1, 1]
S3: [0, 1, 1, 1, 1]


Problems Output Debug Console Terminal Ports
zsh
Actual Execution Time Matrix (tij):
Task 1: [0.00, 0.00, 0.00, 9.00, 8.40]
Task 2: [0.00, 0.00, 8.00, 10.50, 9.00]
Task 3: [0.00, 3.50, 6.40, 8.40, 7.20]
Expense Matrix (eij):
Task 1: [0.00, 0.00, 0.00, 5.40, 5.60]
Task 2: [0.00, 0.00, 6.00, 6.30, 6.00]
Task 3: [0.00, 4.20, 4.80, 5.04, 4.80]
S1 Results:
Max time: 9.00s
Total expense: 11.00€
S2 Results:
Max time: 10.50s
Total expense: 18.30€
S3 Results:
Max time: 8.40s
Total expense: 18.84€
--- Multiplexed Resources ---
R3: 2 users (S2, S3)
R4: 3 users (S1, S2, S3)
R5: 3 users (S1, S2, S3)
### PHASE 1.3: Actual Utility Calculation ###
--- Actual Utility for S1 ---
Time vector: ['0.00', '0.00', '0.00', '9.00', '8.40']
Expense vector: ['0.00', '0.00', '0.00', '5.40', '5.60']
Actual utility: 0.1082
Utility loss: 0.1443
--- Actual Utility for S2 ---
Time vector: ['0.00', '0.00', '8.00', '10.50', '9.00']
Expense vector: ['0.00', '0.00', '6.00', '6.30', '6.00']
Actual utility: 0.0880
Utility loss: 0.0914
--- Actual Utility for S3 ---
Time vector: ['0.00', '3.50', '6.40', '8.40', '7.20']
Expense vector: ['0.00', '4.20', '4.80', '5.04', '4.80']
Actual utility: 0.1059
Utility loss: 0.0928
STEP 1 RESULTS
User Allocation Vector Expected U Actual U Loss
S1 (0, 0, 0, 1, 1) 0.2525 0.1082 0.1443
S2 (0, 0, 1, 1, 1) 0.1794 0.0880 0.0914
S3 (0, 1, 1, 1, 1) 0.1986 0.1059 0.0928
Total Expected Utility: 0.6306
Total Actual Utility: 0.3022
Total Utility Loss: 0.3285
STEP 2: STRATEGY UPDATE WITH UPDATED EXECUTION TIMES

```
### PHASE 2.1: Calculate Updated Execution Times ###


--- Updated Execution Times for Step 2 ---

New Base Execution Times (t̂_new):
  Task 1: [5.00, 5.37, 8.40, 12.30, 11.00]
  Task 2: [6.00, 6.17, 8.80, 12.80, 11.20]
  Task 3: [4.00, 4.67, 8.00, 12.10, 10.60]


### PHASE 2.2: User Re-optimization ###


--- Re-optimizing S1 with updated times ---

--- Optimizing S1 (Step 1) ---
Execution times: [5.0, 5.366666666666667, 8.4, 12.299999999999999, 11.0]
Evaluating 10 possible allocations...
Feasible allocations: 3/10
Optimal allocation: (1, 1, 0, 0, 0)
Expected utility: 0.1641
S1: (1, 1, 0, 0, 0) → utility = 0.1641

--- Re-optimizing S2 with updated times ---

--- Optimizing S2 (Step 1) ---
Execution times: [6.0, 6.166666666666667, 8.8, 12.799999999999999, 11.2]
Evaluating 10 possible allocations...
Feasible allocations: 1/10
Optimal allocation: (1, 1, 1, 0, 0)
Expected utility: 0.0930
S2: (1, 1, 1, 0, 0) → utility = 0.0930

--- Re-optimizing S3 with updated times ---

--- Optimizing S3 (Step 1) ---
Execution times: [4.0, 4.666666666666667, 8.0, 12.099999999999998, 10.6]
Evaluating 5 possible allocations...
WARNING: No feasible allocation found for S3!
Feasible allocations: 0/5
Optimal allocation: (1, 1, 1, 1, 0)
Expected utility: 0.0657
S3: (1, 1, 1, 1, 0) → utility = 0.0657


### PHASE 2.3: Provider Processing (Step 2) ###


--- Processing Allocations ---

Allocation Matrix:
  S1: [1, 1, 0, 0, 0]
  S2: [1, 1, 1, 0, 0]
  S3: [1, 1, 1, 1, 0]

Actual Execution Time Matrix (tij):
  Task 1: [15.00, 12.60, 0.00, 0.00, 0.00]
  Task 2: [18.00, 15.00, 8.00, 0.00, 0.00]
  Task 3: [12.00, 10.50, 6.40, 2.80, 0.00]

Expense Matrix (eij):
  Task 1: [5.00, 5.04, 0.00, 0.00, 0.00]
  Task 2: [6.00, 6.00, 6.00, 0.00, 0.00]
  Task 3: [4.00, 4.20, 4.80, 5.04, 0.00]

S1 Results:
  Max time: 15.00s
  Total expense: 10.04€
```

```
Problems  Output  Debug Console  Terminal  Ports                                                    zsh

--- Multiplexed Resources ---
R1: 3 users (S1, S2, S3)
R2: 3 users (S1, S2, S3)
R3: 2 users (S2, S3)


### PHASE 2.4: Step 2 Actual Utilities ###


--- Actual Utility for S1 ---
Time vector: ['15.00', '12.60', '0.00', '0.00', '0.00']
Expense vector: ['5.00', '5.04', '0.00', '0.00', '0.00']
Actual utility: 0.0694
Utility loss: 0.0946

--- Actual Utility for S2 ---
Time vector: ['18.00', '15.00', '8.00', '0.00', '0.00']
Expense vector: ['6.00', '6.00', '6.00', '0.00', '0.00']
Actual utility: 0.0556
Utility loss: 0.0374

--- Actual Utility for S3 ---
Time vector: ['12.00', '10.50', '6.40', '2.80', '0.00']
Expense vector: ['4.00', '4.20', '4.80', '5.04', '0.00']
Actual utility: 0.0793
Utility loss: -0.0137

======================================================================
 STEP 2 RESULTS
======================================================================

User   Allocation Vector         Expected U   Actual U     Loss
----------------------------------------------------------------------
S1     (1, 1, 0, 0, 0)           0.1641       0.0694       0.0946
S2     (1, 1, 1, 0, 0)           0.0930       0.0556       0.0374
S3     (1, 1, 1, 1, 0)           0.0657       0.0793       -0.0137

----------------------------------------------------------------------
Total Expected Utility: 0.3227
Total Actual Utility:   0.2043
Total Utility Loss:     0.1184

======================================================================
 COMPARISON: STEP 1 vs STEP 2
======================================================================

User   Step 1 Actual U    Step 2 Actual U    Improvement
------------------------------------------------------------
S1     0.1082             0.0694             -0.0388
S2     0.0880             0.0556             -0.0325
S3     0.1059             0.0793             -0.0265

------------------------------------------------------------
Total  0.3022             0.2043             -0.0978

======================================================================
 CONSTRAINT VERIFICATION
======================================================================


Step 1 Constraints:
  S1: Time ✓ (9.00 ≤ 500), Budget ✓ (11.00 ≤ 20)
  S2: Time ✓ (10.50 ≤ 300), Budget ✓ (18.30 ≤ 30)
  S3: Time ✓ (8.40 ≤ 800), Budget ✓ (18.84 ≤ 30)

Step 2 Constraints:
  S1: Time ✓ (15.00 ≤ 500), Budget ✓ (10.04 ≤ 20)
  S2: Time ✓ (18.00 ≤ 300), Budget ✓ (18.00 ≤ 30)
  S3: Time ✓ (12.00 ≤ 800), Budget ✓ (18.04 ≤ 30)
```

# 5.1 Step 1 Results: Independent Optimization

**Step 1 Allocation Table**

| User | Allocation Vector | Expected Utility | Actual Utility | Utility Loss |
|---|---|---|---|---|
| S1 | (0, 0, 0, 1, 1) | 0.2525 | 0.1082 | 0.1443 |
| S2 | (0, 0, 1, 1, 1) | 0.1794 | 0.0880 | 0.0914 |
| S3 | (0, 1, 1, 1, 1) | 0.1986 | 0.1059 | 0.0928 |
| **Total** | - | **0.6306** | **0.3022** | **0.3285** |

**Step 1 Execution Time Matrix (with multiplexing)**

| Task | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| S1 | 0.00 | 0.00 | 0.00 | 9.00 | 8.40 |
| S2 | 0.00 | 0.00 | 8.00 | 10.50 | 9.00 |
| S3 | 0.00 | 3.50 | 6.40 | 8.40 | 7.20 |

**Step 1 Expense Matrix**

| Task | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| S1 | 0.00 | 0.00 | 0.00 | 5.40 | 5.60 |
| S2 | 0.00 | 0.00 | 6.00 | 6.30 | 6.00 |
| S3 | 0.00 | 4.20 | 4.80 | 5.04 | 4.80 |

**Step 1 Resource Multiplexing**

Resources with multiple users:

- R3: 2 users (S2, S3)
- R4: 3 users (S1, S2, S3)

- R5: 3 users (S1, S2, S3)

## 5.2 Step 2 Results: Strategy Update

**Updated Execution Times for Step 2**

Formula: $t_{new\_ij} = \hat{t}_{ij} + \Sigma(tij)/n$

| Task | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| S1 | 5.00 | 5.37 | 8.40 | 12.30 | 11.00 |
| S2 | 6.00 | 6.17 | 8.80 | 12.80 | 11.20 |
| S3 | 4.00 | 4.67 | 8.00 | 12.10 | 10.60 |

**Step 2 Allocation Table**

| User | Allocation Vector | Expected Utility | Actual Utility | Utility Loss |
|---|---|---|---|---|
| S1 | (1, 1, 0, 0, 0) | 0.1641 | 0.0694 | 0.0946 |
| S2 | (1, 1, 1, 0, 0) | 0.0930 | 0.0556 | 0.0374 |
| S3 | (1, 1, 1, 1, 0) | 0.0657 | 0.0793 | -0.0137 |
| **Total** | - | **0.3227** | **0.2043** | **0.1184** |

**Step 2 Execution Time Matrix (with multiplexing)**

| Task | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| S1 | 15.00 | 12.60 | 0.00 | 0.00 | 0.00 |
| S2 | 18.00 | 15.00 | 8.00 | 0.00 | 0.00 |
| S3 | 12.00 | 10.50 | 6.40 | 2.80 | 0.00 |

**Step 2 Expense Matrix**

| Task | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| S1 | 5.00 | 5.04 | 0.00 | 0.00 | 0.00 |
| S2 | 6.00 | 6.00 | 6.00 | 0.00 | 0.00 |
| S3 | 4.00 | 4.20 | 4.80 | 5.04 | 0.00 |

**Step 2 Resource Multiplexing**

Resources with multiple users:

- R1: 3 users (S1, S2, S3)
- R2: 3 users (S1, S2, S3)
- R3: 2 users (S2, S3)

## 5.3 Comparison: Step 1 vs Step 2

| User | Step 1 Actual Utility | Step 2 Actual Utility | Improvement |
|---|---|---|---|
| S1 | 0.1082 | 0.0694 | -0.0388 |
| S2 | 0.0880 | 0.0556 | -0.0325 |
| S3 | 0.1059 | 0.0793 | -0.0265 |
| **Total** | **0.3022** | **0.2043** | **-0.0978** |

## 5.4 Constraint Verification

**Step 1 Constraints**

| User | Max Time | Deadline | Status | Total Expense | Budget | Status |
|---|---|---|---|---|---|---|
| S1 | 9.00 | 500 | ✓ | 11.00 | 20 | ✓ |
| S2 | 10.50 | 300 | ✓ | 18.30 | 30 | ✓ |
| S3 | 8.40 | 800 | ✓ | 18.84 | 30 | ✓ |

**Step 2 Constraints**

| User | Max Time | Deadline | Status | Total Expense | Budget | Status |
|---|---|---|---|---|---|---|
| S1 | 15.00 | 500 | ✓ | 10.04 | 20 | ✓ |
| S2 | 18.00 | 300 | ✓ | 18.00 | 30 | ✓ |
| S3 | 12.00 | 800 | ✓ | 18.04 | 30 | ✓ |

# 6. Analysis

## 6.1 Impact of Resource Multiplexing

In Step 1, users optimized independently without considering that other users might select the same resources. When the Resource Manager calculated actual execution times, multiplexing was discovered:

- **Multiplexed Resources**: R3 (2 users: S2, S3), R4 (3 users: S1, S2, S3), R5 (3 users: S1, S2, S3)
- **Impact**: Execution times increased proportionally to the number of users sharing each resource. For example, R4 had a base execution time of 3.0s for S1, but with 3 users sharing it, the actual execution time became 9.00s (3 × 3.0s).
- **Utility Loss**: Users experienced significant utility loss with an average of 0.1095 per user (total loss: 0.3285). S1 had the largest utility loss (0.1443) due to multiplexing on R4 and R5, which were its chosen resources.

**Specific Impact**:

- S1 chose R4 and R5, both shared by all 3 users, resulting in execution times of 9.00s and 8.40s respectively (3× the base times).
- S2 and S3 both shared R3, R4, and R5, causing their execution times to be multiplied accordingly.
- Despite meeting all constraints, the utility loss demonstrates the importance of considering resource contention in optimization.

## 6.2 Effectiveness of Step 2 Strategy Update

The formula $t\_new_{ij} = \hat{t}_{ij} + \Sigma(tij)/n$ incorporates knowledge from Step 1 into users' optimization for Step 2.

**Observations**:

- **Utility Changes**: None of the users improved their actual utility in Step 2. All three users experienced decreases: S1 (-0.0388), S2 (-0.0325), and S3 (-0.0265). Total system utility decreased from 0.3022 to 0.2043 (-0.0978).
- **Strategy Changes**: All users completely changed their allocation strategies. S1 moved from (0,0,0,1,1) to (1,1,0,0,0), avoiding the heavily multiplexed R4 and R5 in favor of R1 and R2. S2 changed from (0,0,1,1,1) to (1,1,1,0,0), also avoiding R4 and R5. S3 changed from (0,1,1,1,1) to (1,1,1,1,0), avoiding R5.
- **Resource Avoidance**: Resources R4 and R5, which were heavily multiplexed in Step 1 (3 users each), were largely avoided in Step 2. Instead, users shifted to R1 and R2, which became the new bottlenecks with 3 users each.
- **Paradoxical Result**: Despite incorporating information from Step 1, the total utility decreased. This occurred because users' updated strategies created new contention points (R1 and R2 became heavily multiplexed), and the updated execution time formula penalized resources that were previously popular, potentially leading to suboptimal global outcomes.
- **S3 Exception**: S3's expected utility (0.0657) was lower than its actual utility (0.0793), indicating a negative utility loss. This suggests S3 benefited from the actual allocation pattern compared to its expectation, though this may be due to the infeasible allocation warning (no feasible allocation was found, so the optimizer selected the best infeasible option).

## 6.3 Game-Theoretic Insights

This two-step process demonstrates:

1. **Nash Equilibrium Seeking**: Users adjust strategies based on observed outcomes. In Step 2, all users avoided the resources that caused high multiplexing in Step 1 (R4 and R5), demonstrating strategic adaptation.
2. **Strategic Learning**: Step 2 incorporates information from Step 1 through the updated execution time formula. However, this led to all users converging on similar strategies (primarily R1 and R2), creating new bottlenecks.
3. **Fairness vs Efficiency**: The results show a trade-off between individual optimization and system-wide efficiency. While individual users attempted to optimize their utilities based on learned information, the total system utility actually decreased. This illustrates the challenge of reaching a Nash equilibrium where no user can improve by unilaterally changing strategy, but the overall system performance may not be optimal. The constraint satisfaction was maintained for all users in both steps, ensuring feasibility, but efficiency decreased in Step 2.

# 7. Challenges and Solutions

## 7.1 Implementation Challenges

1. **Challenge**: IAM Role Creation Restrictions
    - **Issue**: AWS lab account had restricted IAM permissions, preventing direct creation or modification of IAM roles.
    - **Solution**: Used existing `LabRole` provided by the AWS lab environment. The role already had necessary permissions for DynamoDB, SQS, and Lambda operations. Documented this limitation in the implementation.
2. **Challenge**: DynamoDB Decimal Type Compatibility
    - **Issue**: Lambda functions using `boto3` to store data in DynamoDB encountered errors because DynamoDB doesn't support Python's native float types - it requires Decimal types.
    - **Solution**: Modified `submission_handler.py` to convert float values to `Decimal` type using `Decimal(str(expected_utility))` before storing in DynamoDB.
3. **Challenge**: Lambda Dependencies (requests library)
    - **Issue**: Lambda 2 needed the `requests` library, but it's not included in Python's standard library. Packaging with `pip install` failed due to network restrictions in the deployment environment.
    - **Solution**: Replaced `requests` library with Python's built-in `urllib.request` module, which is available in the Lambda runtime environment without additional dependencies.
4. **Challenge**: Lambda Handler Configuration
    - **Issue**: Lambda functions initially failed with "Unable to import module 'lambda_function'" errors because the handler was set to the default `lambda_function.lambda_handler` instead of the actual module names.
    - **Solution**: Updated Lambda function handler configuration to `submission_handler.lambda_handler` for Lambda 1 and `get_and_send.lambda_handler` for Lambda 2.
5. **Challenge**: DynamoDB Composite Key Updates
    - **Issue**: Lambda 2 failed to update item status in DynamoDB because it was only providing the partition key (`user_id`) but not the sort key (`timestamp`), which is required for composite keys.
    - **Solution**: Modified `get_pending_allocations()` to return both allocations and timestamps, then updated `update_allocation_status()` to use both keys in the DynamoDB update operation.
6. **Challenge**: JSON Serialization of Decimal Types

- **Issue**: DynamoDB returns Decimal types, which cannot be directly serialized to JSON when sending to the Resource Manager.
- **Solution**: Converted Decimal values to integers when retrieving allocation vectors from DynamoDB using list comprehension: `[int(v) for v in allocation_vector]`.

## 7.2 Testing Challenges

1. **Local vs AWS Testing**:
   - **Challenge**: Needed to test locally before deploying to AWS to catch errors early.
   - **Solution**: Developed comprehensive local test suite (`run_tests.py complete`) that simulates the complete flow without AWS dependencies. This allowed validation of optimization algorithms, matrix calculations, and utility computations before deployment.
2. **Debugging Distributed System**:
   - **Challenge**: Errors in distributed systems are harder to debug because components run on different machines and logs are scattered.
   - **Solution**: Used AWS CloudWatch logs for Lambda functions, Flask application logs on EC2 instances, and health check endpoints on all services. Added extensive print statements and error handling to trace the flow through each component.
3. **Network Connectivity Testing**:
   - **Challenge**: Verifying that all EC2 instances could communicate with each other and with AWS services.
   - **Solution**: Used security groups to allow necessary ports (5000, 5001), tested connectivity with curl commands, and verified health endpoints before running the complete flow.
4. **Lambda Function Testing**:
   - **Challenge**: Lambda functions run in a managed environment, making local debugging difficult.
   - **Solution**: Tested Lambda code logic locally with mock events before deployment, then used CloudWatch logs extensively to debug runtime issues. Created minimal deployment packages to reduce complexity.

# 8. Conclusion

This project successfully implemented a game-theoretic resource allocation system demonstrating:

1. **Technical Implementation**: Successfully deployed a distributed system using EC2, Lambda, SQS, and DynamoDB
2. **Optimization Algorithm**: Implemented brute-force optimization with constraint checking
3. **Two-Step Game**: Demonstrated how users can improve strategies with learned information
4. **Resource Multiplexing**: Illustrated real-world challenges of shared resource allocation

**Key Findings**:

- **Step 1 Performance**: Total expected utility was 0.6306, but actual utility after multiplexing was 0.3022, showing a utility loss of 0.3285 (52% reduction) due to resource contention. All constraints were satisfied.
- **Step 2 Performance**: Despite incorporating learned information, total actual utility decreased to 0.2043 (32% decrease from Step 1). This demonstrates that individual strategic adjustments can lead to worse global outcomes, illustrating the complexity of game-theoretic resource allocation.
- **Constraint Satisfaction**: All constraints (deadlines and budgets) were successfully met in both steps for all three users, validating the feasibility of the allocations.
- **Resource Multiplexing Impact**: Resources R4 and R5 were heavily contested in Step 1 (3 users each), leading to 3× execution time multipliers. Users strategically avoided these in Step 2, but this created new bottlenecks on R1 and R2.

# 9. References


1. Wei, G., Vasilakos, A. V., Zheng, Y., & Xiong, N. (2009). A game-theoretic method of fair resource allocation for cloud computing services. The Journal of Supercomputing, 54(2), 252-269.
2. Course Materials: HY452 Project 2025, University of Crete
3. AWS Documentation: EC2, Lambda, DynamoDB, SQS


# Appendix

# A. Complete Code Listings

Attached zip file pdf code with this pdf as well.

# B. AWS Configuration Details

**IAM Roles**:

- **LabRole**: Used by Lambda functions and EC2 instances
    - Note: Due to AWS lab account restrictions, custom IAM roles could not be created. The existing LabRole was used, which provided necessary permissions for DynamoDB, SQS, and Lambda operations.
    - Permissions include: DynamoDB read/write, SQS send/receive, Lambda invoke, EC2 basic operations

**Resource URLs and Identifiers**:

- **SQS Queue**:
    - Name: SubmissionsQueue
    - URL: https://sqs.us-east-1.amazonaws.com/210058569281/SubmissionsQueue
    - Region: us-east-1
- **DynamoDB Table**:
    - Name: ResourceAllocations
    - ARN: arn:aws:dynamodb:us-east-1:210058569281:table/ResourceAllocations
    - Region: us-east-1
    - Keys: Partition Key (user_id), Sort Key (timestamp)
- **EC2 Instances**:
    - User S1: http://18.207.117.128:5000
    - User S2: http://50.19.166.35:5000
    - User S3: http://98.80.12.101:5000

    - Resource Manager: http://52.207.250.246:5001
- **Lambda Functions**:
    - submission-handler: Processes SQS messages, stores in DynamoDB
    - get-and-send: Retrieves from DynamoDB, sends to Resource Manager
    - Region: us-east-1

**Environment Variables**:

Lambda 1 (submission-handler):

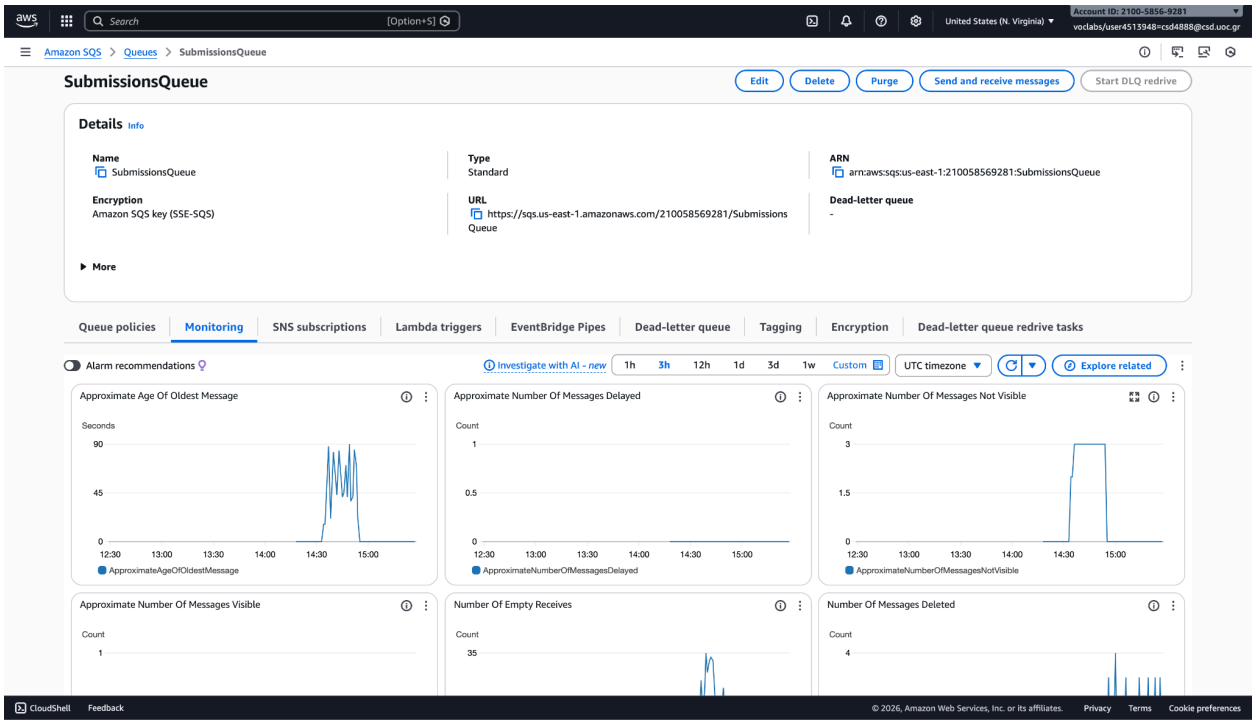
- DYNAMODB_TABLE: ResourceAllocations
- LAMBDA_FUNCTION_2: get-and-send
- AWS_REGION: us-east-1

Lambda 2 (get-and-send):

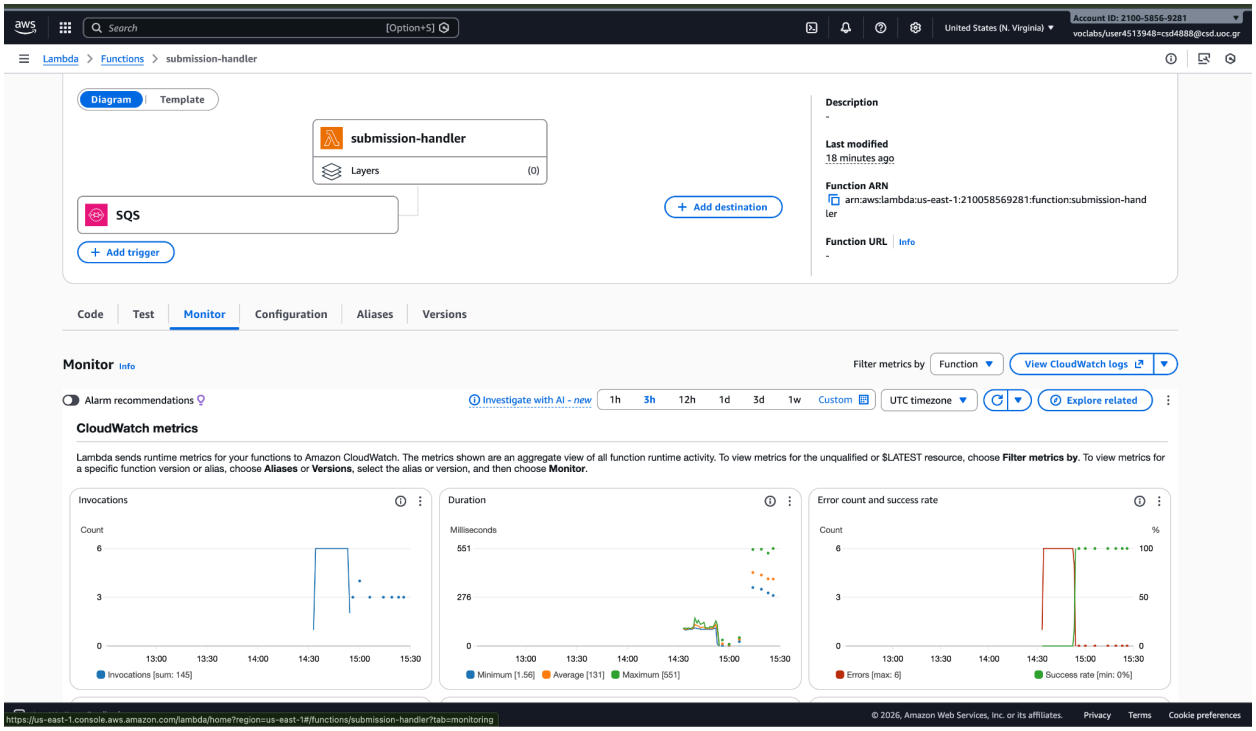
- DYNAMODB_TABLE: ResourceAllocations
- RESOURCE_MANAGER_URL: http://52.207.250.246:5001
- AWS_REGION: us-east-1 (automatically set by Lambda)

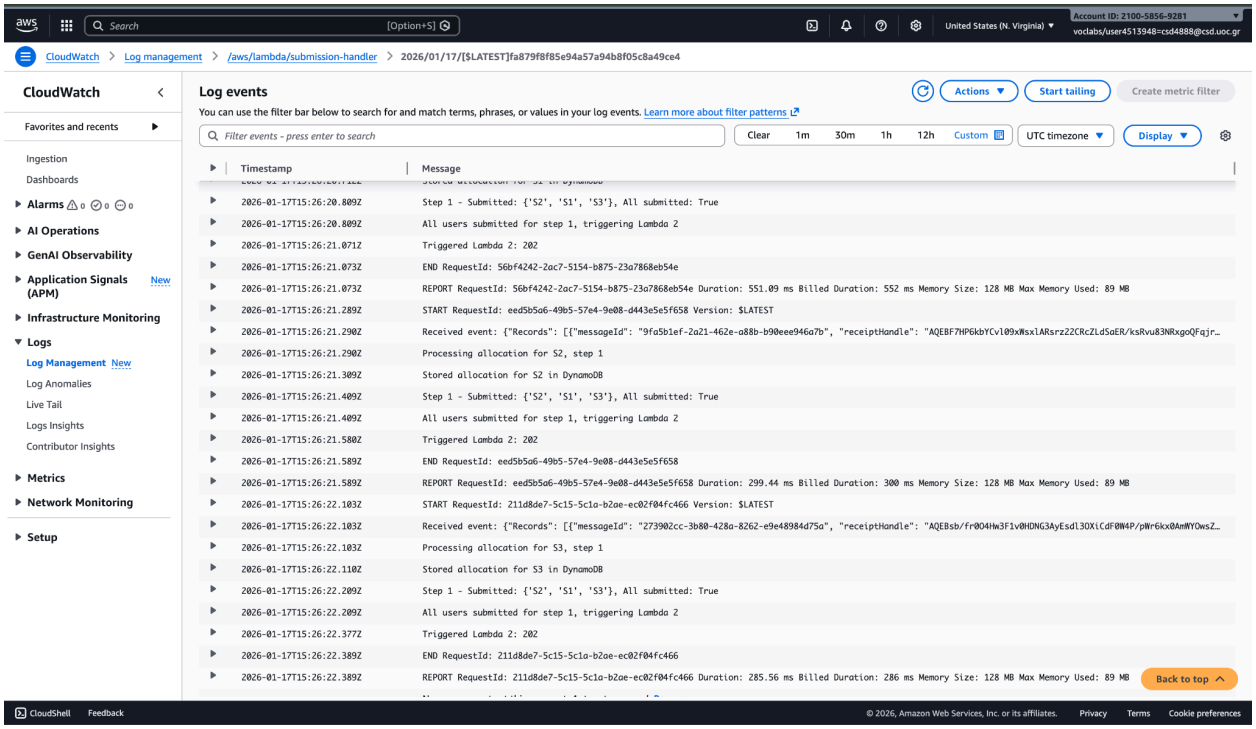
# C. AWS Execution Logs

### SQS QUEUE:

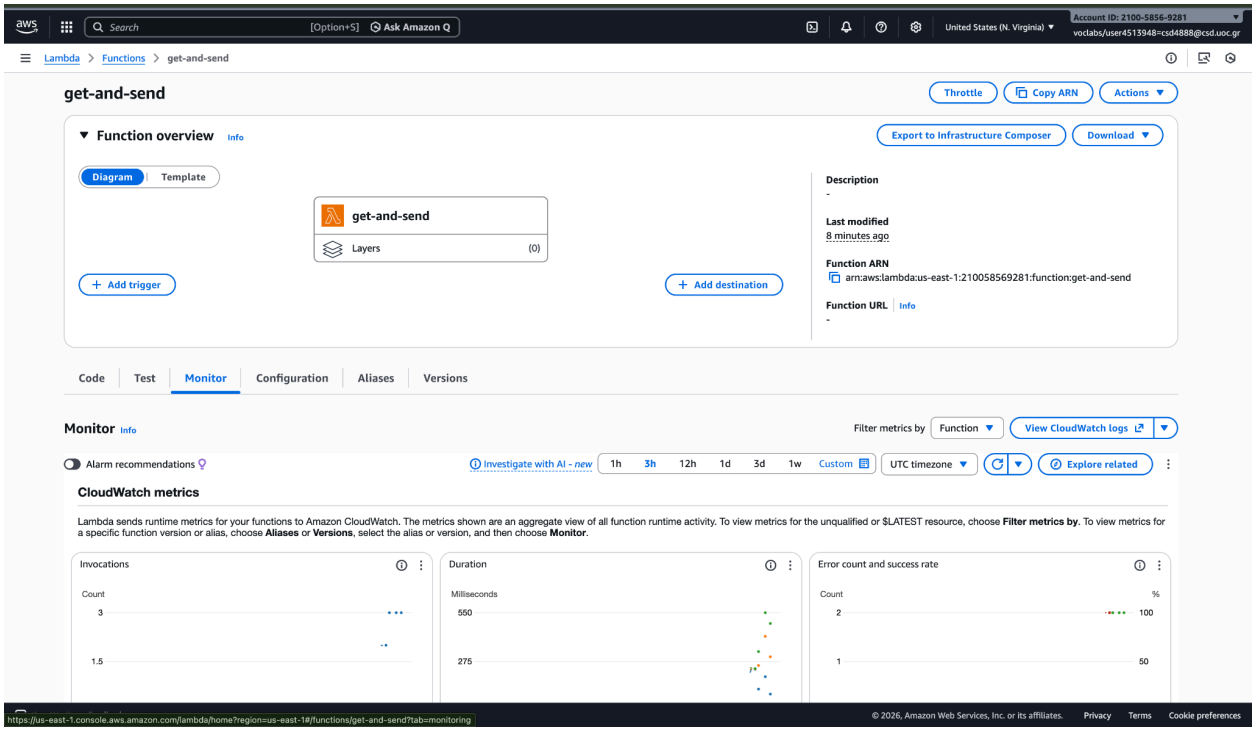


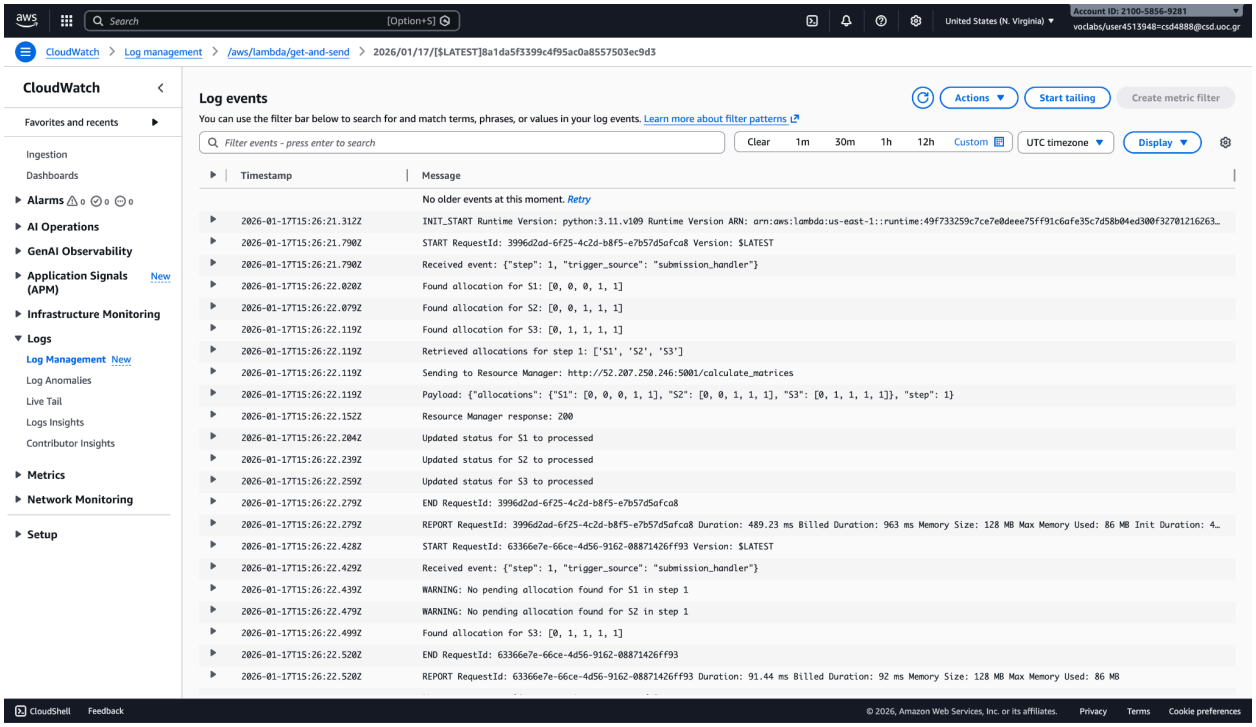
## LAMBDA 1:


Lambda > Functions > submission-handler
Diagram
Template
submission-handler
Layers (0)
SQS
Add trigger
Add destination
Description
-
Last modified
18 minutes ago
Function ARN
arn:aws:lambda:us-east-1:210058569281:function:submission-handler
Function URL Info
-
Code
Test
Monitor
Configuration
Aliases
Versions
Monitor Info
Filter metrics by
Function
View CloudWatch logs
Alarm recommendations
Investigate with AI - new
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
Explore related
CloudWatch metrics
Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose Filter metrics by. To view metrics for a specific function version or alias, choose Aliases or Versions, select the alias or version, and then choose Monitor.
Invocations
Count
Invocations [sum: 145]
Duration
Milliseconds
Minimum [1.56]
Average [131]
Maximum [551]
Error count and success rate
Count
%
Errors [max: 6]
Success rate [min: 0%]
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

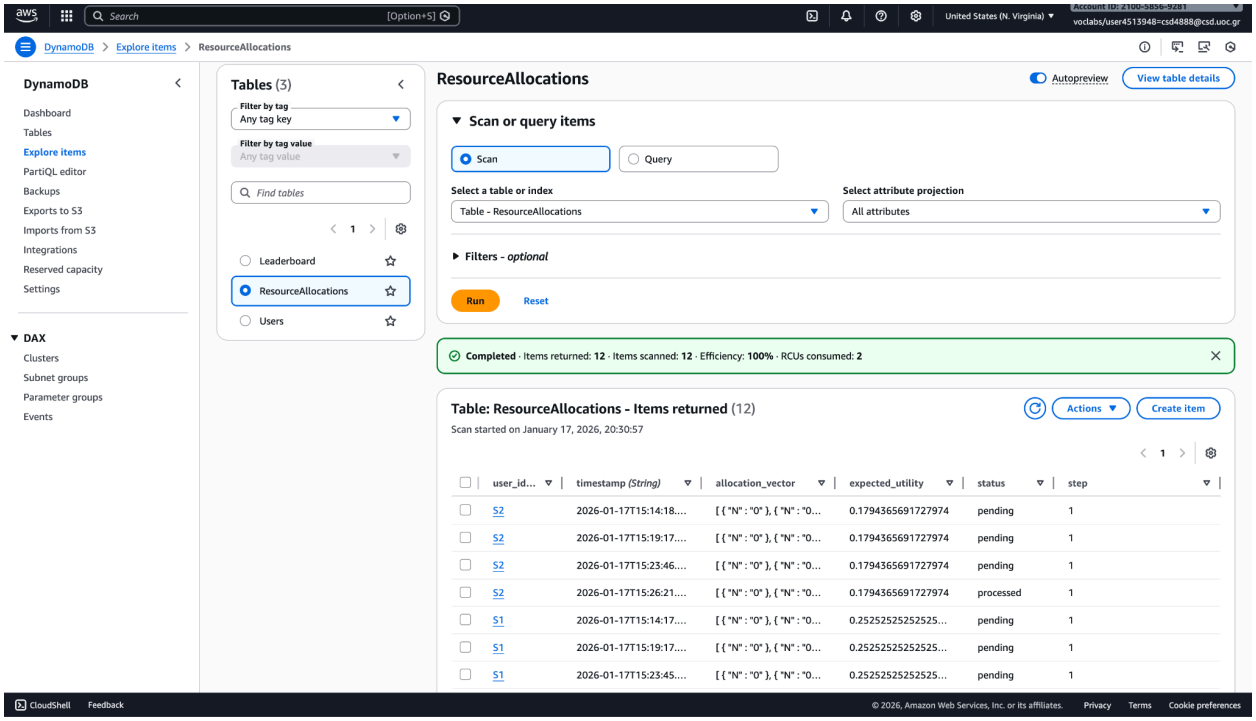

CloudWatch > Log management > /aws/lambda/submission-handler > 2026/01/17/[$LATEST]fa879f8f85e94a57a94b8f05c8a49ce4
CloudWatch
Favorites and recents
Ingestion
Dashboards
Alarms
AI Operations
GenAI Observability
Application Signals (APM) New
Infrastructure Monitoring
Logs
Log Management New
Log Anomalies
Live Tail
Logs Insights
Contributor Insights
Metrics
Network Monitoring
Setup
Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Actions
Start tailing
Create metric filter
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Timestamp
Message
2026-01-17T15:26:20.809Z Step 1 - Submitted: {'S2', 'S1', 'S3'}, All submitted: True
2026-01-17T15:26:20.809Z All users submitted for step 1, triggering Lambda 2
2026-01-17T15:26:21.071Z Triggered Lambda 2: 202
2026-01-17T15:26:21.073Z END RequestId: 56bf4242-2ac7-5154-b875-23a7868eb54e
2026-01-17T15:26:21.073Z REPORT RequestId: 56bf4242-2ac7-5154-b875-23a7868eb54e Duration: 551.09 ms Billed Duration: 552 ms Memory Size: 128 MB Max Memory Used: 89 MB
2026-01-17T15:26:21.289Z START RequestId: eed5b5a6-49b5-57e4-9e08-d443e5e5f658 Version: $LATEST
2026-01-17T15:26:21.290Z Received event: {"Records": [{"messageId": "9fa5b1ef-2a21-462e-a88b-b90eee946a7b", "receiptHandle": "AQEBF7HP6kbYCvl09xWsxlARsrz22CRcZLdSaER/ksRvu83NRxgoQFqjr…
2026-01-17T15:26:21.290Z Processing allocation for S2, step 1
2026-01-17T15:26:21.309Z Stored allocation for S2 in DynamoDB
2026-01-17T15:26:21.409Z Step 1 - Submitted: {'S2', 'S1', 'S3'}, All submitted: True
2026-01-17T15:26:21.409Z All users submitted for step 1, triggering Lambda 2
2026-01-17T15:26:21.580Z Triggered Lambda 2: 202
2026-01-17T15:26:21.589Z END RequestId: eed5b5a6-49b5-57e4-9e08-d443e5e5f658
2026-01-17T15:26:21.589Z REPORT RequestId: eed5b5a6-49b5-57e4-9e08-d443e5e5f658 Duration: 299.44 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 89 MB
2026-01-17T15:26:22.103Z START RequestId: 211d8de7-5c15-5c1a-b2ae-ec02f04fc466 Version: $LATEST
2026-01-17T15:26:22.103Z Received event: {"Records": [{"messageId": "273902cc-3b80-428a-8262-e9e48984d75a", "receiptHandle": "AQEBsb/fr0O4Hw3F1v0HDNG3AyEsdl3OXiCdF0W4P/pWr6kx0AmMYOwsZ…
2026-01-17T15:26:22.103Z Processing allocation for S3, step 1
2026-01-17T15:26:22.110Z Stored allocation for S3 in DynamoDB
2026-01-17T15:26:22.209Z Step 1 - Submitted: {'S2', 'S1', 'S3'}, All submitted: True
2026-01-17T15:26:22.209Z All users submitted for step 1, triggering Lambda 2
2026-01-17T15:26:22.377Z Triggered Lambda 2: 202
2026-01-17T15:26:22.389Z END RequestId: 211d8de7-5c15-5c1a-b2ae-ec02f04fc466
2026-01-17T15:26:22.389Z REPORT RequestId: 211d8de7-5c15-5c1a-b2ae-ec02f04fc466 Duration: 285.56 ms Billed Duration: 286 ms Memory Size: 128 MB Max Memory Used: 89 MB
Back to top
CloudShell
Feedback
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

## LAMBDA 2:


Lambda > Functions > get-and-send
get-and-send
Throttle
Copy ARN
Actions
Function overview
Export to Infrastructure Composer
Download
Diagram
Template
get-and-send
Layers (0)
+ Add trigger
+ Add destination
Description
-
Last modified
8 minutes ago
Function ARN
arn:aws:lambda:us-east-1:210058569281:function:get-and-send
Function URL
-
Code
Test
Monitor
Configuration
Aliases
Versions
Monitor
Filter metrics by
Function
View CloudWatch logs
Alarm recommendations
Investigate with AI - new
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
Explore related
CloudWatch metrics
Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose Filter metrics by. To view metrics for a specific function version or alias, choose Aliases or Versions, select the alias or version, and then choose Monitor.
Invocations
Count
3
1.5
Duration
Milliseconds
550
275
Error count and success rate
Count
%
2
1
100
50
https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/get-and-send?tab=monitoring
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


CloudWatch > Log management > /aws/lambda/get-and-send > 2026/01/17/[$LATEST]8a1da5f3399c4f95ac0a8557503ec9d3
CloudWatch
Favorites and recents
Ingestion
Dashboards
Alarms
AI Operations
GenAI Observability
Application Signals (APM)
New
Infrastructure Monitoring
Logs
Log Management
New
Log Anomalies
Live Tail
Logs Insights
Contributor Insights
Metrics
Network Monitoring
Setup
Log events
You can use the filter bar below to search for and match terms, phrases, or values in your log events.
Learn more about filter patterns
Actions
Start tailing
Create metric filter
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display
Timestamp
Message
No older events at this moment.
Retry
2026-01-17T15:26:21.312Z INIT_START Runtime Version: python:3.11.v109 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:49f733259c7ce7e0deee75ff91c6afe35c7d58b04ed300f32701216263…
2026-01-17T15:26:21.790Z START RequestId: 3996d2ad-6f25-4c2d-b8f5-e7b57d5afca8 Version: $LATEST
2026-01-17T15:26:21.790Z Received event: {"step": 1, "trigger_source": "submission_handler"}
2026-01-17T15:26:22.020Z Found allocation for S1: [0, 0, 0, 1, 1]
2026-01-17T15:26:22.079Z Found allocation for S2: [0, 0, 1, 1, 1]
2026-01-17T15:26:22.119Z Found allocation for S3: [0, 1, 1, 1, 1]
2026-01-17T15:26:22.119Z Retrieved allocations for step 1: ['S1', 'S2', 'S3']
2026-01-17T15:26:22.119Z Sending to Resource Manager: http://52.207.250.246:5001/calculate_matrices
2026-01-17T15:26:22.119Z Payload: {"allocations": {"S1": [0, 0, 0, 1, 1], "S2": [0, 0, 1, 1, 1], "S3": [0, 1, 1, 1, 1]}, "step": 1}
2026-01-17T15:26:22.152Z Resource Manager response: 200
2026-01-17T15:26:22.204Z Updated status for S1 to processed
2026-01-17T15:26:22.239Z Updated status for S2 to processed
2026-01-17T15:26:22.259Z Updated status for S3 to processed
2026-01-17T15:26:22.279Z END RequestId: 3996d2ad-6f25-4c2d-b8f5-e7b57d5afca8
2026-01-17T15:26:22.279Z REPORT RequestId: 3996d2ad-6f25-4c2d-b8f5-e7b57d5afca8 Duration: 489.23 ms Billed Duration: 963 ms Memory Size: 128 MB Max Memory Used: 86 MB Init Duration: 4…
2026-01-17T15:26:22.428Z START RequestId: 63366e7e-66ce-4d56-9162-08871426ff93 Version: $LATEST
2026-01-17T15:26:22.429Z Received event: {"step": 1, "trigger_source": "submission_handler"}
2026-01-17T15:26:22.439Z WARNING: No pending allocation found for S1 in step 1
2026-01-17T15:26:22.479Z WARNING: No pending allocation found for S2 in step 1
2026-01-17T15:26:22.499Z Found allocation for S3: [0, 1, 1, 1, 1]
2026-01-17T15:26:22.520Z END RequestId: 63366e7e-66ce-4d56-9162-08871426ff93
2026-01-17T15:26:22.520Z REPORT RequestId: 63366e7e-66ce-4d56-9162-08871426ff93 Duration: 91.44 ms Billed Duration: 92 ms Memory Size: 128 MB Max Memory Used: 86 MB
CloudShell
Feedback
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

## DYNAMODB:


DynamoDB > Explore items > ResourceAllocations
DynamoDB
Dashboard
Tables
Explore items
PartiQL editor
Backups
Exports to S3
Imports from S3
Integrations
Reserved capacity
Settings
DAX
Clusters
Subnet groups
Parameter groups
Events
Tables (3)
Filter by tag
Any tag key
Filter by tag value
Any tag value
Find tables
Leaderboard
ResourceAllocations
Users
ResourceAllocations
Autopreview
View table details
Scan or query items
Scan
Query
Select a table or index
Table - ResourceAllocations
Select attribute projection
All attributes
Filters - optional
Run
Reset
Completed · Items returned: 12 · Items scanned: 12 · Efficiency: 100% · RCUs consumed: 2
Table: ResourceAllocations - Items returned (12)
Scan started on January 17, 2026, 20:30:57
Actions
Create item
user_id... | timestamp (String) | allocation_vector | expected_utility | status | step
S2 | 2026-01-17T15:14:18.... | [ { "N" : "0" }, { "N" : "0... | 0.1794365691727974 | pending | 1
S2 | 2026-01-17T15:19:17.... | [ { "N" : "0" }, { "N" : "0... | 0.1794365691727974 | pending | 1
S2 | 2026-01-17T15:23:46.... | [ { "N" : "0" }, { "N" : "0... | 0.1794365691727974 | pending | 1
S2 | 2026-01-17T15:26:21.... | [ { "N" : "0" }, { "N" : "0... | 0.1794365691727974 | processed | 1
S1 | 2026-01-17T15:14:17.... | [ { "N" : "0" }, { "N" : "0... | 0.25252525252525... | pending | 1
S1 | 2026-01-17T15:19:17.... | [ { "N" : "0" }, { "N" : "0... | 0.25252525252525... | pending | 1
S1 | 2026-01-17T15:23:45.... | [ { "N" : "0" }, { "N" : "0... | 0.25252525252525... | pending | 1
CloudShell
Feedback
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences