National College of Ireland

# Optimising Serverless Data Ingestion: Evaluating the Impact of AWS Lambda Configuration on Performance and Cost in Amazon S3 Data Lakes

MSc Research Project
Cloud Computing

School of Computing
National College of Ireland

Abhishek Selvadurai
Student ID: 23416882

Supervisor: Ahmed Makki

Submission Date: 11-12-2025

**National College of Ireland**
**Student Declaration**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Abhishek Selvadurai |
| **Student ID:** | 23416882 |
| **Programme:** | Cloud Computing |
| **Year:** | 2025-2026 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Ahmed Makki |
| **Submission Date:** | 11-12-2025 |
| **Project Title:** | Optimising Serverless Data Ingestion: Evaluating the Impact of AWS Lambda Configuration on Performance and Cost in Amazon S3 Data Lakes |
| **Word Count:** | 3701 |
| **Page Count:** | 16 |

I hereby certify that the information contained in this configuration manual is accurate and complete. All instructions, commands, and procedures have been tested and verified. Any references to external sources, tools, or documentation are properly cited in the bibliography section.

I understand that:

- All internet material and external sources must be referenced in the bibliography section
- Students are required to use the Referencing Standard specified in the report template
- Using other authors' written or electronic work without proper citation is illegal (plagiarism) and may result in disciplinary action
- This configuration manual represents my own work and understanding of the system

| | |
|---|---|
| **Signature:** | Abhishek Selvadurai |
| **Date:** | 11-12-2025 |

# Configuration Manual

Abhishek Selvadurai

23416882

## 1 System Requirements

This section defines what kind of hardware and software are required to run this project, Optimizing Serverless Data Ingesting: Measure Performance and Cost Effects of AWS Lambda Configuration on Data Lake Storage using Amazon S3, on both local systems as well as using AWS.

| **Component** | **Minimum Requirement** | **Recommended** |
|---|---|---|
| CPU | 2 cores | 4–8 cores |
| Memory | Minimum of 4 GB system memory | 8 GB of system memory |
| Disk Capacity | At least 500 MB of available disk capacity | Preferably 1 GB or greater storage space |
| Network | Stable internet connection | High-speed broadband connection |

Table 1: Hardware Requirements

| **Software** | **Minimum Requirement** | **Recommended** |
|---|---|---|
| System Platform | Ubuntu version 20.04 or later, macOS 10.15 or later, Windows 10/11 with WSL2 | Latest stable OS version |
| Python | Python 3.11+ | Latest Python 3.x release |
| Terraform | Version 1.0+ | Latest stable Terraform version |
| AWS CLI | AWS CLI v2.x | Most recent AWS CLI v2.x build |
| Git | Installed (optional) | Latest stable Git version |
| ZIP Utility | Native OS ZIP tool | Updated ZIP/archiving utility |

Table 2: Software Requirements

### 1.1 Python Dependencies

All Python dependencies are provided in the project's `requirements.txt` Notable libraries include:

- **Boto3** for interacting with AWS Lambda, S3, and CloudWatch services.
- **Pandas** (optional) for data analysis and CSV processing.

These packages are installed automatically during the installation steps in Section 2.

### 1.2 AWS Account and IAM Requirements

To deploy and run experiments on AWS, an active AWS account is required. The project requires an IAM role with Lambda execution permissions. The role name is specified in `infra/terraform.tfvars` and must be created before deployment. The IAM Role shall be provided with the permission set defined below:

- **CloudWatch Logs**: permissions for `CreateLogGroup`, `CreateLogStream`, and `PutLogEvents`.
- **S3**: `PutObject`, `GetObject`, `ListBucket` permissions for the experiment output bucket.
- **CloudWatch Metrics**: allow PutMetricData permission (related to Embedded Metric Format).

This role is associated with all the Lambda functions during deployment through Terraform.

### 1.3 AWS Service Quotas

Verify that the AWS account has sufficient service quotas.

- **Lambda Functions**: 54 functions; the default limit is 1,000.
- **Lambda Concurrent Executions**: Depends on the reserved concurrency setting; its default value is 1,000.
- **S3 Buckets**: 1 bucket (default limit: 100).
- **CloudWatch Log Groups**: There are currently 54 log groups, with a default limit of 500.

## 2 Installation Guide

This chapter describes the setup needed to run the Lambda S3 Ingestion Performance Experiment. Deployment using CloudShell has the advantage of very limited preparatory work. The other approach, which will also be covered, is local deployment.

**Method 1 (CloudShell Deployment) – Recommended:**

1. Upload the project ZIP archive into AWS CloudShell.
2. Extract the archive and then change into the project directory in CloudShell.
3. Create an IAM role in the AWS Management Console, with the appropriate permissions, that will be used for Lambda execution.

4. Fill `infra/terraform.tfvars` with the values for the project.
5. Run the deployment script `deploy_eu_west.sh` to provision all the resources.
6. Verify successful deployment by checking the Lambda functions, S3 bucket, and CloudWatch log groups.

**Method 2 (Local Deployment) – Alternative:**

1. Extract the project archive and open the respective project folder.
2. AWS CLI should be installed and configured with proper credentials using the command aws configure.
3. Install Terraform.
4. Create an IAM role for Lambda execution, including all the permissions required.
5. Fill `infra/terraform.tfvars` with the values for the project.
6. Run the deployment script `deploy_eu_west.sh` to provision all the resources.
7. Verify the deployment by checking the Lambda functions, S3 bucket, and the CloudWatch log groups.

**Note:** AWS CloudShell comes pre-configured with AWS CLI credentials, Terraform, and Python; hence, it eliminates the need for local installation and configuration processes.

## 2.1 Step 1: Upload and Extract Project in CloudShell

**CloudShell Deployment (Recommended):**

1. The top navigation bar of the AWS Console contains the CloudShell icon. Click this icon to launch your AWS CloudShell session.
2. Upload the project ZIP file to CloudShell:
   - Click the *Actions* menu (three dots) in CloudShell.
   - Select *Upload file*.
   - Choose `lambda-factorial-exp.zip` from the local machine.
   - Wait for the upload to complete.
3. Extract the ZIP file in CloudShell:

```
unzip lambda-factorial-exp.zip
```

4. Navigate to the project directory:

```
cd lambda-factorial-exp
```

5. Verify the project structure:

```
ls -la lambda_app/ infra/ runner/ deploy_eu_west.sh
```

**Alternative: Local Deployment**

If deploying locally instead of CloudShell, extract the project archive and navigate to the project directory:

**Linux/macOS:**

```
unzip lambda-factorial-exp.zip
cd lambda-factorial-exp
```

**Windows (Git Bash or WSL):**

```
unzip lambda-factorial-exp.zip
cd lambda-factorial-exp
```

**Windows (PowerShell):**

```
Expand-Archive -Path lambda-factorial-exp.zip -DestinationPath .
cd lambda-factorial-exp
```

Verify the project structure:

```
ls -la lambda_app/ infra/ runner/ deploy_eu_west.sh
```

## 2.2 Step 2: Install and Configure AWS CLI (Local Deployment Only)

**Note:** This step is only required for local deployment (Method 2). CloudShell comes pre-configured with AWS CLI credentials, so skip this step if using CloudShell.

Install AWS CLI based on the operating system:

**Linux:**

```
curl -L "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o ↩
↪ awscliv2.zip
unzip ./awscliv2.zip
sudo ./aws/install --update
```

**macOS:**

```
# Install AWS CLI via Homebrew
brew install awscli
```

**Windows:** You can obtain the AWS CLI MSI installation file by visiting the AWS website and downloading it to your local computer. You can then install it on your computer.

Configure AWS CLI with credentials:

```
aws configure   # configure AWS credentials
```

Please enter all of these items upon request.

- Your AWS Access Key ID
- Your AWS Secret Access Key

- The default AWS region (for example: eu-west-1)
- The preferred output format (such as json)

To confirm that the setup is correct, run:

```
aws sts  get-caller-identity   # verify identity
```

## 2.3 Step 3: Install Terraform (Local Deployment Only)

**Note:** This step is only required for local deployment (Method 2). CloudShell comes with Terraform pre-installed, so skip this step if using CloudShell.

**Linux/macOS:**

```
# Using Homebrew (macOS) or package manager (Linux)
brew install terraform  # macOS
# or
sudo apt-get update    # update package index
sudo apt-get install terraform   # install terraform
```

**Windows:** Download and install Terraform from HashiCorp's website. Ensure that Terraform is added to the Path in your Windows environment variables.

To confirm the setup:

```
terraform   --version   # check version
```

## 2.4 Step 4: Create IAM Execution Role

Create a new IAM role for Lambda execution that has appropriate permissions:

1. To create a new role in AWS, log in to the AWS Management Console and select IAM. Go to the Roles section.
2. *Create a new role* by clicking the Create role button. Select *AWS service* for the type of trusted entity and choose *Lambda* for the purpose of the role.
3. Assign a name to the role (e.g., `lambda-execution-role`).
4. Attach the required policies. Based on the final configuration used in this experiment (see Figure X), the role includes the following permissions:
   - `AWSLambdaBasicExecutionRole` – provides essential permissions for Lambda to write logs to CloudWatch.
   - `CloudWatchLambdaInsightsExecutionRolePolicy` – enables enhanced monitoring via CloudWatch Lambda Insights.
   - `AmazonS3FullAccess` –Grants the ability to read and write to S3 buckets via S3 API calls, which allow experiement to store data in S3 buckets.
   - `lambda-cloudwatch-logs` (inline policy) – custom policy enabling additional log group and log stream operations.
5. Complete the role creation process and save the role.

6. Record the role name, as it is required later in the `terraform.tfvars` file for deploying the Lambda functions.

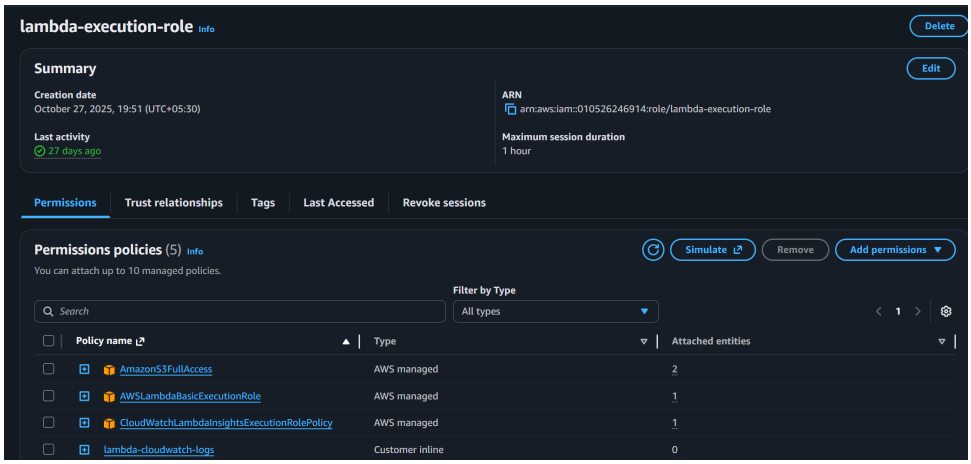


Figure 1: Creating the IAM execution role for Lambda functions with required policies.

## 2.5 Step 5: Configure Terraform Variables

Edit `infra/terraform.tfvars` with specific values:

```
region = "eu-west-1"
environment = "lab"
lambda_execution_role_name = "lambda-execution-role"
output_bucket_name = "lambda-s3-exp-lab-23102002-789456"
event_bytes_default = 1024
object_mb_default = 100
memory_levels_mb = [512, 1024, 2048]
events_batch_levels = [1, 10, 100]
multipart_mb_levels = [8, 32, 64]
reserved_concurrency_lvls = [0, 10, 30]
project_name = "lambda-s3-experiment-lab"
```

**Important:**

- Change the name of your `Lambda_Execution_Role` to be the actual IAM Role that you created in step 4.
- The output bucket must be a globally unique Amazon S3 bucket that is different from any existing Amazon S3 bucket in your account.
- Adjust other values as needed for the experiment configuration.

## 2.6 Step 6: Run Deployment Script

### 2.6.1 CloudShell Deployment (Recommended)

After completing Steps 1, 4, and 5 in CloudShell, set execution permissions and run the deployment script:

```
chmod +x deploy_eu_west.sh
./deploy_eu_west.sh
```

The script performs the following operations:

- Cleans up existing Lambda functions with prefix "ingest-"
- Deletes existing S3 buckets matching the pattern
- Deletes existing CloudWatch log groups
- Initializes Terraform
- Plans the deployment
- Applies the deployment (creates 54 Lambda functions, S3 bucket, CloudWatch log groups)
- Outputs deployment information

**Note:** CloudShell comes pre-configured with AWS CLI credentials, so no additional `aws configure` step is required. CloudShell also includes Terraform, Python, and other common utilities pre-installed.

### 2.6.2 Local Deployment (Alternative)

This method requires AWS CLI to be configured locally using `aws configure` (completed in Step 2).

Set execution permissions for the deployment script (Linux/macOS):

```
chmod +x deploy_eu_west.sh
```

Execute the deployment script from the project root:

**Linux/macOS:**

```
./deploy_eu_west.sh
```

**Windows (Git Bash or WSL):**

```
bash deploy_eu_west.sh
```

The script performs the same operations as CloudShell deployment:

- Cleans up existing Lambda functions with prefix "ingest-"
- Deletes existing S3 buckets matching the pattern
- Deletes existing CloudWatch log groups
- Initializes Terraform
- Plans the deployment
- Applies the deployment (creates 54 Lambda functions, S3 bucket, CloudWatch log groups)
- Outputs deployment information

## 2.7 Step 7: Verify Deployment

Verify that all resources were created successfully:

```
# View Terraform outputs
cd infra
terraform output

# Verify Lambda functions
aws lambda list-functions  --region eu-west-1 \
  --query "Functions[?starts_with(FunctionName, 'ingest-')].FunctionName" \
  --output table

# Verify S3 bucket
aws s3 ls | grep lambda-s3-exp

# Verify CloudWatch log groups
aws logs describe-log-groups --region eu-west-1 \
  --log-group-name-prefix "/aws/lambda/ingest-" \
  --query "logGroups[].logGroupName" --output table
```

## 2.8 Optional: Local Development Setup

For local development and testing, set up the project on a local machine:

1. Clone or extract the repository:

```
unzip lambda-factorial-exp.zip
cd lambda-factorial-exp
```

2. To create a virtual environment type the following in your command line:

```
python3   -m   venv  venv
source  venv/bin/activate    # Windows: .\venv\Scripts\Activate.ps1
```

3. Install the required packages:

```
python3 -m pip install -r requirements.txt
```

4. Run a local test (if applicable):

```
python -m runner.experiment_runner --help
```

At this point, the system is fully configured and ready to run experiments. Proceed to Section 3 for instructions on executing experiments and verifying results.

# 3 Usage Guide

This section explains how to run experiments using the deployed Lambda functions and verify that results have been stored in Amazon S3 and logged to CloudWatch.

## 3.1 Running Experiments

After completing all installation steps in Section 2, the system is ready to run experiments using the experiment runner script.

### 3.1.1 Running Events Workload Experiments

Run experiments for the events workload:

```
python -m runner.experiment_runner \
    --function-prefix "ingest-events-" \
    --invocations 100 \
    --trials 5 \
    --region eu-west-1
```

This command will:

- Discover all functions starting with "ingest-events-" (27 functions)
- Run 5 trials per function
- Each trial invokes the function 100 times
- Collect results and save to CSV file

### 3.1.2 Running Batch Workload Experiments

Run experiments for the batch workload:

```
python -m runner.experiment_runner \
    --function-prefix "ingest-batch-" \
    --invocations 100 \
    --trials 5 \
    --region eu-west-1
```

**Note:** Batch workload functions take longer (up to 15 minutes timeout), so use fewer invocations per trial.

### 3.1.3 Experiment Parameters

The experiment runner accepts the following parameters:

- `--function-prefix`: Prefix to match Lambda function names ("ingest-events-" or "ingest-batch-")
- `--invocations`: Number of invocations per trial (Events: 50-100, Batch: 20-100)
- `--trials`: Number of trials per function (5-10 trials recommended for statistical significance)
- `--region`: AWS region where functions are deployed (default: eu-west-1)
- `--output`: Output CSV filename (auto-generated with timestamp if not specified)

## 3.2 Verifying Results in Amazon S3

After the experiment runner completes execution, verify that results have been successfully saved to the S3 bucket:

1. Log in to AWS Console and Click on S3 Service
2. Locate the bucket configured in `terraform.tfvars` (the `output_bucket_name` value).
3. Navigate through the bucket structure to find experiment outputs.
4. Confirm that objects have been uploaded with the expected naming pattern:
   - Events workload: `events/<run_id>/<timestamp>-<uuid>.jsonl`
   - Batch workload: `batch/<run_id>/<timestamp>-<uuid>.jsonl`

If all files are present, the experiment has completed successfully and results have been uploaded to S3. These files can be downloaded for local analysis if needed.

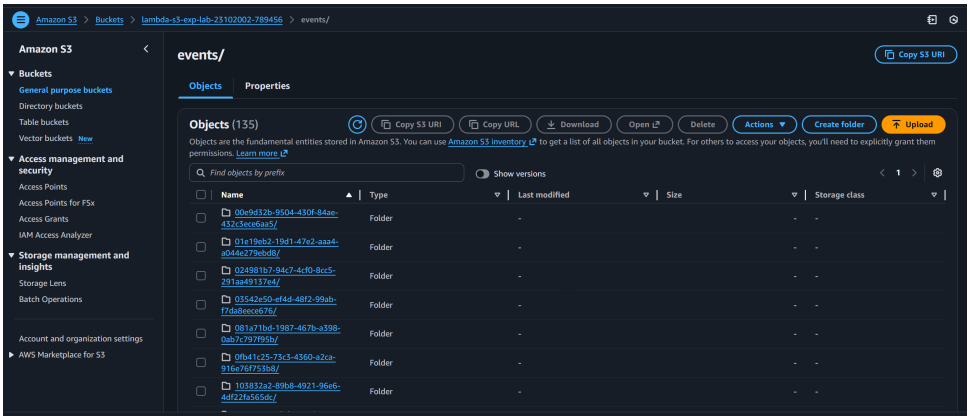


Figure 2: Uploaded experiment artefacts in the S3 bucket.

## 3.3 Checking CloudWatch Logs

To verify that logs have been created in CloudWatch and to monitor the execution:

1. Login to your AWS Console, and search for the *CloudWatch* Service on the search bar of the Console.
2. To view *log groups*, click on *Logs* in the Navigation pane, then select Log Groups. You will see a list of all the available log groups you may view and filter based on your requirements.
3. Locate log groups with the prefix `/aws/lambda/ingest-` (one log group per Lambda function).
4. Click on a log group to view log streams.
5. Open the most recent log stream to review the execution logs.
6. Verify that the following information is present:

- Function invocation start and completion messages
- CloudWatch Embedded Metric Format (EMF) metric logs
- S3 upload status and object keys
- Run ID and execution metadata
- Cold start detection and timing information
- Any errors or warnings that occurred during execution

The CloudWatch logs provide detailed information about the experiment execution, making it easy to troubleshoot issues and monitor progress. Logs are written in real-time, so execution can be monitored as it happens.

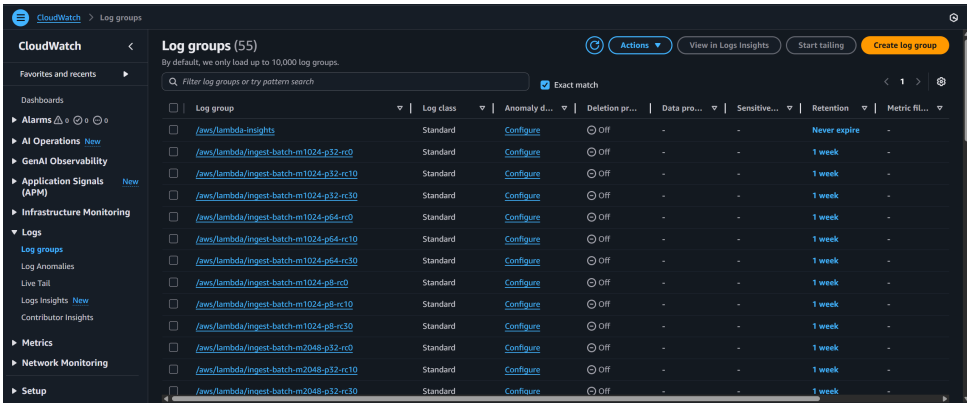


Figure 3: CloudWatch log groups and streams for Lambda functions.

## 3.4 Checking CloudWatch Metrics

In addition to logs, the system also publishes custom metrics to CloudWatch using Embedded Metric Format (EMF):

1. In the CloudWatch console, navigate to *Metrics* → *All metrics*.
2. Find the `LambdaS3Study` namespace under *Custom namespaces*.
3. Review the available metrics, which include:
   - `latency_ms` – total execution latency in milliseconds
   - `object_bytes` – size of uploaded object in bytes
   - `events_generated` – number of events generated (events workload)
   - `cold_start_ms` – cold start initialization time in milliseconds
4. Metrics are dimensioned by: `workload`, `function_name`, `region`, `run_id`
5. Develop custom dashboards or alerts from these, as needed.

These metrics characterize the performance of systems and may serve either monitoring or analytics objectives.

## 3.5 Metrics Gathering

This subsection defines the steps for gathering and then analyzing the CloudWatch metrics that will be part of the Lambda S3 Ingestion Performance Experiment.

### 3.5.1 Prerequisites

Install required utilities before running metrics collection scripts:

**Linux environments (Amazon Linux / RHEL):**

```
sudo yum install -y bc jq
```

**Linux (Ubuntu/Debian):**

```
sudo apt-get install -y bc jq
```

**macOS:**

```
brew install bc jq
```

### 3.5.2 Configuration Variables

To activate metric collection, define the variables listed below, before data collection:

```
# Specify the region, the bucket name, the prefix applied in the naming ↩
↪ convention for S3 keys, and the time window over which to fetch the ↩
↪ CloudWatch metrics.
REGION=eu-west-1
BUCKET=<bucket-name>
PREFIX=ingest
END="$(date -u +%FT%TZ)"
START="$(date -u -d '16 hours ago' +%FT%TZ)"
START_EPOCH="$(date -u -d "$START" +%s)"
END_EPOCH="$(date -u -d "$END" +%s)"
WINDOW_SECS="$((END_EPOCH-START_EPOCH))"
```

**Important:** Replace `<bucket-name>` with the actual S3 bucket name used for experiment outputs.

### 3.5.3 Function Discovery

Identify all Lambda functions matching the experiment prefix:

```
FUNCS="$(aws lambda list-functions --region "$REGION" \
  --query "Functions[?starts_with(FunctionName, ↩
↪ \`$PREFIX\`)].FunctionName" \
  --output text)"

printf '%s\n' $FUNCS
```

### 3.5.4 Collecting Lambda Metrics

Collect Lambda performance metrics for each function:

```
# Collect Lambda p95 duration, invocations, memory, billed duration, and ↩
↪ GB-seconds
TOTAL_GBSECONDS=0
```

```
TOTAL_INVOCATIONS=0

for fn in $FUNCS; do
  echo "Function: $fn"

  # p95 execution duration
  p95_duration=$(aws cloudwatch get-metric-statistics --region "$REGION" \
    --namespace AWS/Lambda --metric-name Duration \
    --dimensions Name=FunctionName,Value="$fn" \
    --statistics Average --extended-statistics p95 \
    --start-time "$START" --end-time "$END" --period 60 \
    | jq -r '[.Datapoints[].ExtendedStatistics."p95"] | (if length>0 then (add/length) else 0 end)')

  # Total invocations
  invocations=$(aws cloudwatch get-metric-statistics --region "$REGION" \
    --namespace AWS/Lambda --metric-name Invocations \
    --dimensions Name=FunctionName,Value="$fn" \
    --statistics Sum --start-time "$START" --end-time "$END" --period 60 \
    | jq '[.Datapoints[].Sum] | add // 0')

  # Memory configuration
  memory_mb=$(aws lambda get-function-configuration --region "$REGION" \
    --function-name "$fn" --query 'MemorySize' --output text)

  # Total execution duration (ms)
  duration_sum_ms=$(aws cloudwatch get-metric-statistics --region "$REGION" \
    --namespace AWS/Lambda --metric-name Duration \
    --dimensions Name=FunctionName,Value="$fn" \
    --statistics Sum --start-time "$START" --end-time "$END" --period 60 \
    | jq '[.Datapoints[].Sum] | add // 0')

  # Convert duration + memory  GB-seconds
  gb_seconds=$(MEM_MB="$memory_mb" DUR_MS="$duration_sum_ms" python3 - <<'PY'
import os, decimal
D=decimal.Decimal
dur_ms=D(os.environ.get("DUR_MS","0") or "0")
mem_mb=D(os.environ.get("MEM_MB","0") or "0")
gbs=(dur_ms/D("1000"))*(mem_mb/D("1024"))
print(f"{gbs:.6f}")
PY
)

  # Accumulate totals
  TOTAL_GBSECONDS=$(echo "$TOTAL_GBSECONDS + $gb_seconds" | bc)
  TOTAL_INVOCATIONS=$(echo "$TOTAL_INVOCATIONS + $invocations" | bc)
done
```

### 3.5.5 Collecting S3 Metrics

Collect S3 ingestion metrics:

```
# Collect S3 ingestion metrics
PUTS_TOTAL=$(aws cloudwatch get-metric-statistics --region "$REGION" \
  --namespace AWS/S3 --metric-name PutRequests \
  --dimensions Name=BucketName,Value="$BUCKET"
```

```
↪ Name=FilterId,Value=EntireBucket \
  --statistics Sum --start-time "$START" --end-time "$END" --period 60 \
  | jq '[.Datapoints[].Sum] | add // 0')

BYTES_UPLOADED=$(aws cloudwatch get-metric-statistics --region "$REGION" \
  --namespace AWS/S3 --metric-name BytesUploaded \
  --dimensions Name=BucketName,Value="$BUCKET" ↩
↪ Name=FilterId,Value=EntireBucket \
  --statistics Sum --start-time "$START" --end-time "$END" --period 60 \
  | jq '[.Datapoints[].Sum] | add // 0')
```

### 3.5.6 Retrieving AWS Pricing Data

Retrieve current AWS pricing data for cost calculation:

```
# Retrieve AWS pricing data
LAM_ALL=$(aws pricing get-products --region us-east-1 --service-code ↩
↪ AWSLambda --output json)
S3_ALL=$(aws pricing get-products --region us-east-1 --service-code ↩
↪ AmazonS3 --output json)
```

**Note:** AWS Pricing API is only available in the `us-east-1` region, regardless of where resources are deployed.

### 3.5.7 Cost Calculation

Calculate final costs for Lambda and S3 operations:

```
# Extract pricing values (adjust based on current AWS pricing)
PRICE_REQ_PER_1M=0.20      # Lambda requests per 1M invocations
PRICE_GBSECOND=0.0000166667 # Lambda GB-seconds
PRICE_PUT_PER_1K=0.005     # S3 PUT requests per 1K requests

# Final cost computation for Lambda + S3 operations
lambda_cost_usd=$(awk -v inv="$TOTAL_INVOCATIONS" -v ↩
↪ gbs="$TOTAL_GBSECONDS" \
  -v pr1m="$PRICE_REQ_PER_1M" -v pgbs="$PRICE_GBSECOND" \
  'BEGIN{printf "%.10f",(inv/1000000.0)*pr1m + gbs*pgbs}')

s3_ops_cost_usd=$(awk -v puts="$PUTS_TOTAL" -v ppk="$PRICE_PUT_PER_1K" \
  'BEGIN{printf "%.10f",(puts/1000.0)*ppk}')

total_cost_usd=$(awk -v a="$lambda_cost_usd" -v b="$s3_ops_cost_usd" \
  'BEGIN{printf "%.10f",a+b}')

echo "Lambda Cost: \$$lambda_cost_usd"
echo "S3 Operations Cost: \$$s3_ops_cost_usd"
echo "Total Cost: \$$total_cost_usd"
```

### 3.5.8 Collecting Custom EMF Metrics

Query custom metrics from the `LambdaS3Study` namespace:

```
# List all custom metrics in LambdaS3Study namespace
aws cloudwatch list-metrics \
    --namespace LambdaS3Study \
```

```
    --region "$REGION"

# Get latency metrics
aws cloudwatch get-metric-statistics \
    --namespace LambdaS3Study \
    --metric-name latency_ms \
    --dimensions Name=workload,Value=events ↩
↪ Name=function_name,Value=ingest-events-m512-b1-rc0 \
    --start-time "$START" \
    --end-time "$END" \
    --period 300 \
    --statistics Average,Maximum,Minimum,Sum,SampleCount \
    --region "$REGION"

# Get cold start metrics
aws cloudwatch get-metric-statistics \
    --namespace LambdaS3Study \
    --metric-name cold_start_ms \
    --dimensions Name=workload,Value=events \
    --start-time "$START" \
    --end-time "$END" \
    --period 300 \
    --statistics Average,Maximum,Minimum \
    --region "$REGION"

# Get object bytes metrics
aws cloudwatch get-metric-statistics \
    --namespace LambdaS3Study \
    --metric-name object_bytes \
    --dimensions Name=workload,Value=batch \
    --start-time "$START" \
    --end-time "$END" \
    --period 300 \
    --statistics Average,Maximum,Minimum,Sum \
    --region "$REGION"
```

### 3.5.9 Exporting Metrics for Analysis

Export metrics to JSON files for further analysis:

```
# Export Lambda duration metrics
aws cloudwatch get-metric-statistics \
    --namespace AWS/Lambda \
    --metric-name Duration \
    --dimensions Name=FunctionName,Value=ingest-events-m512-b1-rc0 \
    --start-time "$START" \
    --end-time "$END" \
    --period 300 \
    --statistics Average,Maximum,Minimum \
    --region "$REGION" > metrics_duration.json

# Export custom latency metrics
aws cloudwatch get-metric-statistics \
    --namespace LambdaS3Study \
    --metric-name latency_ms \
    --dimensions Name=workload,Value=events \
    --start-time "$START" \
```

```
    --end-time "$END" \
    --period 3600 \
    --statistics Average,Maximum,Minimum \
    --region "$REGION" > metrics_latency.json
```

## 3.6 Troubleshooting

If results are not appearing in S3 or logs are not being created in CloudWatch:

- Verify that the IAM execution role is correctly attached to all Lambda functions and has the required policies.
- Check that the S3 bucket name in `terraform.tfvars` is correct and accessible.
- Ensure that the CloudWatch log groups exist or can be created by the IAM role.
- Review the terminal output for any error messages during script execution.
- Check the CloudWatch logs for detailed error information.
- Confirm Lambda functions have been deployed and are accessible.

```
aws lambda list-functions --region eu-west-1 \
    --query 'Functions[?starts_with(FunctionName, ↩
↪ `ingest-`)].FunctionName'
```

- Perform manual testing of a single function call:

```
aws lambda invoke \
    --function-name ingest-events-m512-b1-rc0 \
    --payload '{"run_id":"test-run-123"}' \
    --region eu-west-1 \
    response.json
```