

## AWS-Lambda- Assignment1

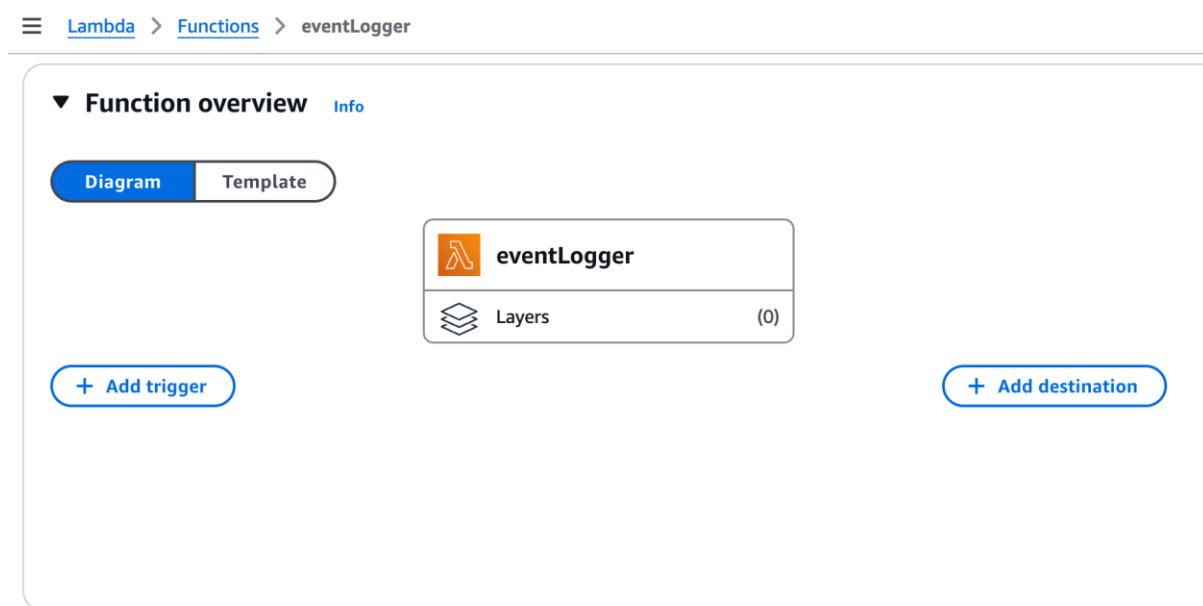
### SHASHANK BG

#### Lab tasks

##### Task 1 - Navigate the AWS Lambda Console

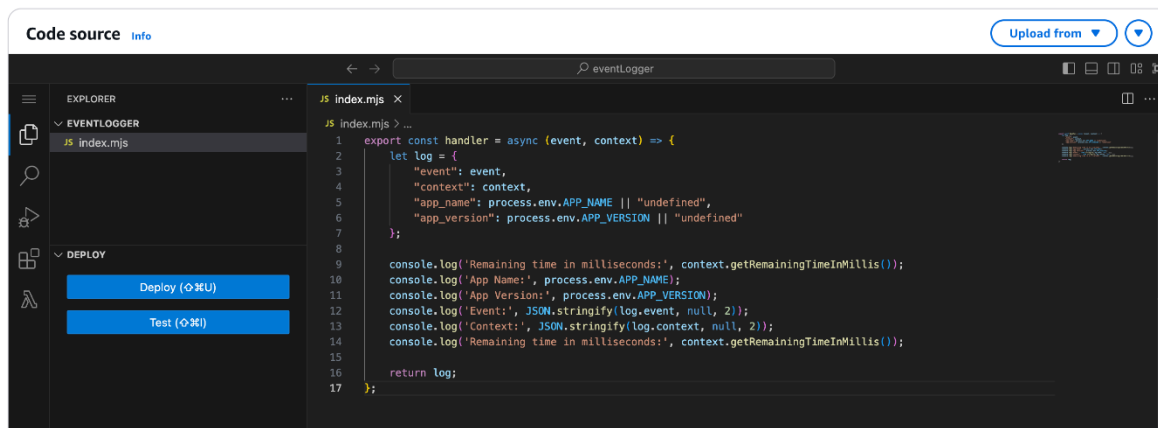
This task will introduce you to the AWS Lambda console. You will explore key sections of the Lambda function configuration, including runtimes, handlers, environment variables, execution roles, and basic settings. By the end of this task, you will understand how to navigate the Lambda console and modify function settings effectively.

- Open the **AWS Management Console** and navigate to **AWS Lambda**.
- From the **Functions** section, open the function eventLogger.
- In the **Function overview** section, you'll see the Diagram section that displays the Layers at the center, and source event triggers on the left, and destinations on the right.



- Click on **Add trigger** to explore different ways Lambda functions can be invoked. Here you can choose and configure one or more triggers for your function. Triggers can be synchronous, like **API Gateway** and **ALB**, allowing real-time execution, or asynchronous, like **S3** and **SNS**, where events fire the function automatically. Poll-based triggers, such as **SQS** and **DynamoDB Streams**, allow Lambda to pull data when available. Understanding triggers is crucial for designing event-driven architectures.





- The **Code source** panel provides you with a web-based IDE to write, test and deploy your code. This also provides option to use CloudWatch Live Tail feature.

```
1. export const handler = async (event, context) => {
2.   let log = {
3.     "event": event,
4.     "context": context,
5.     "app_name": process.env.APP_NAME || "undefined",
6.     "app_version": process.env.APP_VERSION || "undefined"
7.   };
8.
9.   console.log('Remaining time in milliseconds:', context.getRemainingTimeInMillis());
10.  console.log('App Name:', process.env.APP_NAME);
11.  console.log('App Version:', process.env.APP_VERSION);
12.  console.log('Event:', JSON.stringify(log.event, null, 2));
13.  console.log('Context:', JSON.stringify(log.context, null, 2));
14.  console.log('Remaining time in milliseconds:', context.getRemainingTimeInMillis());
15.
16.  return log;
17. };
```

- Scroll down to review **Code properties**, **Runtime settings** and **Layers**. Explore and review different configuration options available.

Lambda > Functions > eventLogger

ENVIRONMENT VARIABLES

Package size  
448 byte

SHA256 hash  
3PXD2QdmWoN3f27KombPDPupGhhggDxUKAjPZsyUmk=

Last modified  
15 minutes ago

Encryption with AWS KMS customer managed KMS key

Runtime settings

Runtime  
Node.js 22.x

Handler  
index.handler

Architecture  
x86\_64

Runtime management configuration

Layers

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
There is no data to display.					

- Review the **Test** tab. You can create and run test events here to simulate different inputs for your Lambda function. AWS provides predefined test event templates such as API Gateway request, S3 event, or SNS notification. You can modify these templates or create custom events in JSON format. Running test events helps validate function logic before deploying to production.

Lambda > Functions > eventLogger

Code **Test** Monitor Configuration Aliases Versions

Test event

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

Create new event Edit saved event

Event name

MyEventName

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

Private

Shareable

Template - optional

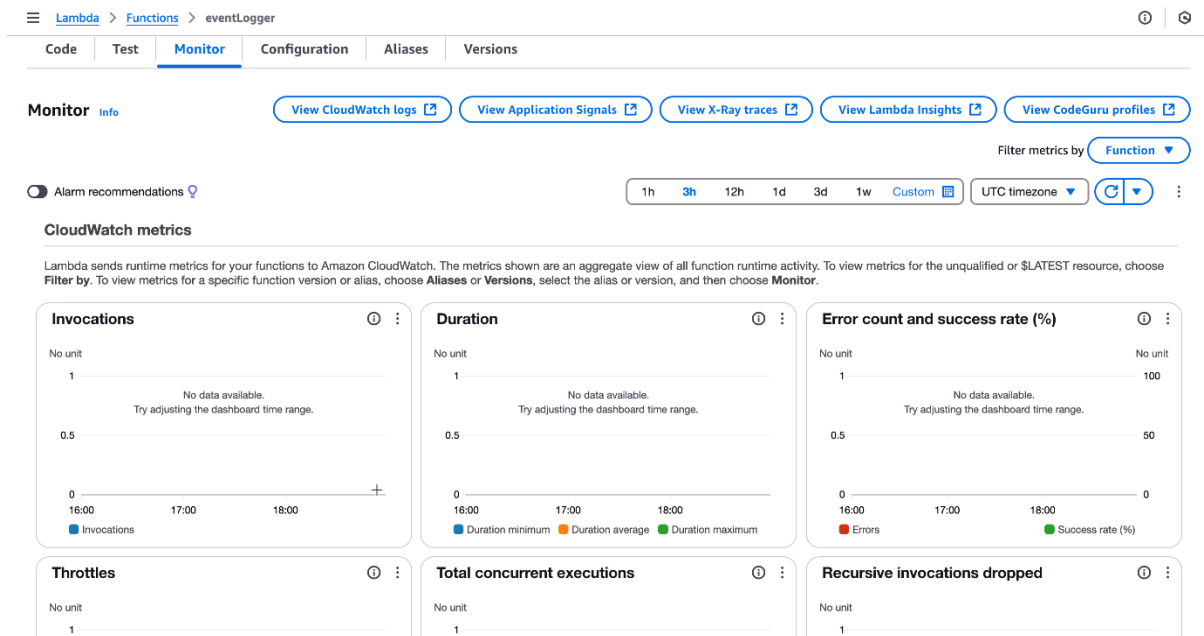
hello-world

Event JSON

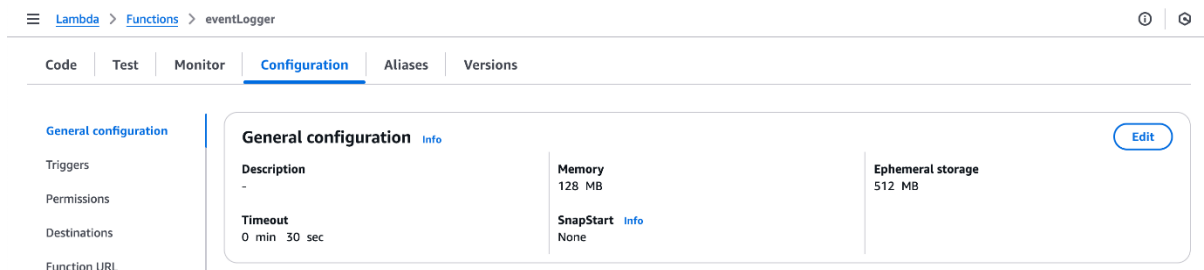
```
1 {
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3"
5 }
```

Format JSON

- Review the **Monitor** tab. This section provides essential monitoring and logging tools:



- **CloudWatch Logs:** View detailed execution logs, errors, and debugging information.
- **X-Ray Traces:** Analyze request flow, latency, and function dependencies.
- **Lambda Insights:** Monitor function execution metrics like duration, memory usage, error rates, and invocation counts.
- These monitoring tools help optimize function performance and identify bottlenecks.
- Review the **Aliases** and **Versions** tabs. The **Versions** tab allows you to create immutable versions of your Lambda function for tracking changes over time. The **Aliases** tab helps manage version deployments by assigning friendly names to specific function versions, making it easier to route traffic between different versions in production environments.
- Open the **Configuration** tab. This section provides access to multiple configuration options that control the behavior, security, and execution of your Lambda function.
  - **General configuration:** This section allows you to configure basic settings such as memory allocation, timeout duration, SnapStart (for supported runtimes), and execution role. These settings impact function performance and execution limits. SnapStart allow you to improve cold start performance. Click the **Edit** button to review all settings available.



- **Triggers:** Here, you can add and manage event sources that invoke the function. This includes configuring event-driven invocations using services such as API Gateway, S3, DynamoDB Streams, and SQS.

- **Permissions:** This section displays IAM permissions assigned to the function. The **lab-lambda-execution-role** includes AWSLambdaBasicExecutionRole policy providing the function CloudWatch logging access.

The screenshot displays the AWS Lambda console's Configuration tab for a function named 'eventLogger'. The left-hand navigation pane includes links for General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, and File systems. The main content area is titled 'Execution role' and features a red arrow pointing to the 'Role name' 'lab-lambda-execution-role'. Below this, the 'Resource summary' section provides a dropdown menu currently set to 'Amazon CloudWatch Logs'. The 'By resource' tab is selected, showing a table of permissions. The table has two columns: 'Resource' and 'Actions'. Under 'All resources', the actions listed are 'Allow: logs:CreateLogGroup', 'Allow: logs:CreateLogStream', and 'Allow: logs:PutLogEvents'. A final note indicates that the information was obtained from the 'Managed policy AWSLambdaBasicExecutionRole, statement 0'.

- **Destinations:** This section allows you to specify where execution results should be sent, such as SQS, SNS, another Lambda function, or an S3 bucket.
- **Function URL:** Here, you can set up a dedicated HTTPS endpoint to invoke the function directly without requiring an API Gateway integration.
- **Environment variables:** This section allows you to define key-value pairs that customize function behavior dynamically without modifying the code. These are useful for configuration settings, API keys, or feature toggles.
- **Tags:** This section enables you to assign metadata labels to organize, categorize, and track Lambda functions for cost allocation and resource grouping.
- **VPC:** Here, you can configure the function to access private resources within an Amazon Virtual Private Cloud (VPC), such as RDS databases, private APIs, and other internal services.
- **RDS databases:** This section allows you to establish secure connectivity between Lambda and Amazon RDS instances to enable database operations within the function.
- **Monitoring and operations tools:** This section enables integration with CloudWatch alarms, AWS AppConfig, and Lambda Insights to track function performance and set up alerts.

Navigation: Lambda > Functions > eventLogger

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

**Monitoring and operations tools**

Concurrency and recursion detection

### Logging configuration Info Edit

CloudWatch log group [/aws/lambda/eventLogger](#)

Log format: Text

### Additional monitoring tools Info Edit

<b>Logs and metrics (default)</b> Enabled	<b>CloudWatch Application Signals and AWS X-Ray</b> Application Signals: Not enabled Lambda service traces: Not enabled	<b>Lambda Insights enhanced monitoring</b> Not enabled	<b>CodeGuru code profiling</b> Not enabled
--	---	---	---

### Extensions

Use extensions to integrate existing tools with your Lambda functions. Visit the [Extensions page](#) to learn about the available AWS partner extensions.

- Concurrency and recursion detection:** Here, you can set reserved concurrency limits to prevent excessive invocations. This is useful for managing function scaling and ensuring predictable resource utilization.

Navigation: Lambda > Functions > eventLogger

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

**Monitoring and operations tools**

**Concurrency and recursion detection**

Asynchronous invocation

Code signing

File systems

State machines

### Concurrency Edit

**Function concurrency**  
Use unreserved account concurrency

**Unreserved account concurrency**  
10

### Provisioned concurrency configurations Edit Remove Add

To enable your function to scale without fluctuations in latency, use provisioned concurrency. You can use Application Auto Scaling to automatically adjust provisioned concurrency to maintain a configured target utilization. Provisioned concurrency runs continually and has separate pricing for concurrency and execution duration. [Learn more](#)

Find configuration

Qualifier	Type	Provisioned concurrency	Status	Details
No configurations				

Add configuration

### Recursive loop detection Info Edit

Recursive loop detection automatically detects and stops infinite recursive loops involving your functions and supported AWS services. This feature is free for all customers.

**Recursion detection configuration**  
Terminate recursive loops

- Asynchronous invocation:** This section allows you to configure retry policies, dead-letter queues (DLQ), and error handling mechanisms for functions that process asynchronous events.

## Edit asynchronous configuration

**Retries** Info

**Maximum age of event**  
The maximum amount of time to keep unprocessed events in the queue.

6 h 0 min 0 sec

**Retry attempts**  
The maximum number of times to retry when the function returns an error.

2

**Dead-letter queue** Info

**Dead-letter queue service**  
You can send unprocessed events from an asynchronous invocation to an Amazon SQS queue or an Amazon SNS topic.

None

- None ✓
- Amazon SNS
- Amazon SQS

Cancel Save

- **Code signing:** This section enables integrity validation to ensure that only signed and trusted function code is deployed.
- **File systems:** Here, you can attach an Amazon Elastic File System (EFS) to provide persistent storage access for the function across multiple invocations.
- **State machines:** This section allows you to integrate Lambda with AWS Step Functions to design and manage complex workflows that involve multiple execution steps.

By completing this task, you now have a solid understanding of the AWS Lambda console. You should be comfortable navigating function settings, configuring triggers and destinations, testing functions, monitoring performance, and exploring advanced configurations.

### Task 2 - Modify Memory, Timeout, Environment Variables, and VPC Settings

This task will guide you through adjusting memory allocation, function timeout, and environment variables. Optimizing these settings improves function performance and allows for better configuration management. Additionally, if your function needs access to resources inside a VPC, you will configure it to run inside **lab-vpc** with private subnets and an appropriate security group.

#### Adjust Memory and Timeout

Memory and timeout settings impact the function's performance and execution limits. Increasing memory provides additional CPU power, while adjusting timeout ensures the function has sufficient time to complete its execution.

- Open the **Configuration** tab. And click **Edit** under **General configuration**.
- Adjust **Memory** from **128 MB** to **512 MB** to allocate more resources to the function. Increasing memory enhances performance and provides more CPU power, as memory and CPU are proportionally linked in AWS Lambda.
- Change the **Timeout** from **30 seconds** to **1 Minute**. A longer timeout ensures the function has enough time to complete tasks that involve processing large datasets, interacting with external services, or performing complex operations.



## Edit basic settings

**Basic settings** [Info](#)

Description - optional

**Memory** [Info](#)  
Your function is allocated CPU proportional to the memory configured.

512

 MB  
Set memory to between 128 MB and 10240 MB

**Ephemeral storage** [Info](#)  
You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#) [↗](#)

512

 MB  
Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

**SnapStart** [Info](#)  
Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the [SnapStart compatibility considerations](#) [↗](#). For Python and .NET runtimes, [view pricing](#) [↗](#).

None

  
Supported runtimes: .NET 8 (C#/.NET/PowerShell), Java 11, Java 17, Java 21, Python 3.12, Python 3.13.

**Timeout**

1

 min 

0

 sec

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#) [↗](#).  
☒ 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.

lab-lambda-execution-role

[View the lab-lambda-execution-role role](#) [↗](#) on the IAM console.

- Click **Save** to apply these changes.

## Add Environment Variables

Environment variables provide a way to pass configuration settings dynamically without modifying the function code. These variables can store application settings, feature flags, or secrets used by the function.

- Navigate to the **Environment variables** section.
- Click **Edit** and then **Add environment variable**.
- Add the following key-value pairs:
  - APP\_NAME = lab-app
  - APP\_VERSION = v1
- Click **Save** to apply the environment variables.

[Lambda](#) > [Functions](#) > [eventLogger](#) > Edit environment variables

**Edit environment variables**

**Environment variables**  
You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#) [↗](#)

Key	Value	
APP_NAME	lab-app	<a href="#">Remove</a>
APP_VERSION	v1	<a href="#">Remove</a>

[Add environment variable](#)

**Encryption configuration**

[Cancel](#) [Save](#)

## Configure VPC Settings

By default, AWS Lambda functions run in an isolated environment without direct access to private resources inside a Virtual Private Cloud (VPC). To allow this function to connect securely to internal services such as **Amazon RDS, EC2 instances, private API endpoints, or message queues**, it needs to be configured to run inside a **VPC**.

Note that this configuration is only needed for accessing resources that reside in a VPC. If you do not need to access any VPC resources, this configuration is not required.

For this exercise, let's assume that this function needs to access a VPC resource such as an Amazon Elastic File System (EFS) volume. To support this access, we will configure the Lambda function to run inside lab-vpc, selecting appropriate private subnets and a security group.

- Scroll to **VPC** section and click **Edit**.
- From the **VPC dropdown**, select lab-vpc. This allows the function to operate within the private network.
- Choose **two private subnets** from lab-vpc. Private subnets ensure the function remains secure by preventing direct internet access while allowing it to communicate with other internal services such as **EFS, RDS, and internal APIs**.

≡ [Lambda](#) > [Functions](#) > [eventLogger](#) > Edit VPC ⓘ ⓘ

### Edit VPC

**VPC**

ⓘ When you connect a function to a VPC in your account, it does not have access to the internet unless your VPC provides access. To give your function access to the internet, route outbound traffic to a NAT gateway in a public subnet. [Learn more](#)

**VPC Info**

Choose a VPC for your function to access.

vpc-09722b7ec3f30a8bf (10.0.0.0/16) ⓘ

☐ Allow IPv6 traffic for dual-stack subnets  
You can allow outbound IPv6 traffic to subnets that have both IPv4 and IPv6 CIDR blocks.

**Subnets**

Select the VPC subnets for Lambda to use to set up your VPC configuration.

Choose subnets ⓘ

subnet-00c79ee62430452fe (10.0.3.0/24) us-east-1c ⓘ  
aws:cloudformation:logical-id: PrivateSubnet2  
aws:cloudformation:stack-id: arn:aws:cloudformation:us-east-1:767397810700:stack/c124162a404688919552383t1w767397810700/38e2f3e0-0104-11f0-bf92-12ff1ca58a87  
aws:cloudformation:stack-name: c124162a404688919552383t1w767397810700 cloudlab: c124162a404688919552383t1w767397810700  
Name: lab-private-subnet-2

subnet-0db6b449b45ad0f90 (10.0.2.0/24) us-east-1b ⓘ  
aws:cloudformation:logical-id: PrivateSubnet1  
aws:cloudformation:stack-id: arn:aws:cloudformation:us-east-1:767397810700:stack/c124162a404688919552383t1w767397810700/38e2f3e0-0104-11f0-bf92-12ff1ca58a87  
aws:cloudformation:stack-name: c124162a404688919552383t1w767397810700 cloudlab: c124162a404688919552383t1w767397810700  
Name: lab-private-subnet-1

You must select between 1 and 16 Subnets.

- Note that the VPC must have a NAT Gateway attached if this function needs to access the internet.
- Select the **security group** lab-sg-private to permit the necessary inbound and outbound traffic for accessing the VPC resources.

**Security groups**  
Choose the VPC security groups for Lambda to use to set up your VPC configuration. The table below shows the inbound and outbound rules for the security groups that you choose.

Choose security groups

**sg-01a92acab61fcdffb1 (lab-sg-private)**

Allow outbound access for Lambda

aws:cloudformation:logical-id: LambdaSecurityGroup

aws:cloudformation:stack-id: arn:aws:cloudformation:us-east-1:767397810700:stack/c124162a404688919552383t1w767397810700/38e2f3e0-0104-11f0-bf92-12ff1ca58a87

aws:cloudformation:stack-name: c124162a404688919552383t1w767397810700 cloudlab: c124162a404688919552383t1w767397810700

You must select between 1 and 5 Security groups.

**Inbound rules** | Outbound rules

Security group ID	Protocol	Ports	Source
sg-01a92acab61fcdffb1	All TCP	0 - 65535	sg-01a92acab61fcdffb1

Cancel Save

Click **Save** to apply the VPC configuration.

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

**VPC**

RDS databases

Monitoring and operations tools

Concurrency and recursion detection

**VPC info**

VPC  
vpc-09722b7ec3f30a8bf (10.0.0.0/16) | lab-vpc

**Subnets**

- Allow IPv6 traffic = false
- subnet-00c79ee62430452fe (10.0.3.0/24) | us-east-1c, lab-private-subnet-2
- subnet-0db6b449b45ad0f90 (10.0.2.0/24) | us-east-1b, lab-private-subnet-1

**Security groups**

- sg-01a92acab61fcdffb1 (lab-sg-private)

**Inbound rules** | Outbound rules

Security group ID	Protocol	Ports	Source
sg-01a92acab61fcdffb1	All TCP	0 - 65535	sg-01a92acab61fcdffb1

With these configurations in place, the Lambda function now has optimized execution settings, dynamically managed environment variables, and secure access to private network resources inside lab-vpc.

### Task 3 - Explore Lambda Security Model

AWS Lambda uses **IAM roles** and **resource-based policies** to securely control access to AWS services and external resources. This task will help you understand how Lambda interacts with AWS services using execution roles and function policies.

By default, every Lambda function runs with an **execution role** that grants it permission to interact with other AWS services. This role is defined in **IAM (Identity and Access Management)** and attached to the function. Additionally, when services like **Amazon S3**, **API Gateway**, or **EventBridge** need to invoke Lambda, a **resource-based policy** is automatically created to define which services or accounts are allowed to trigger the function.

#### Review Execution Role and Add Permissions

Each Lambda function has an **execution role** that controls its permissions. This role allows the function to interact with AWS services securely without using static credentials.

- In the **Permissions** section of the Lambda configuration, locate **Execution role** and click on the role name. This will redirect you to the IAM console.

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration  
Triggers  
**Permissions**  
Destinations  
Function URL  
Environment variables  
Tags  
VPC  
RDS databases  
Monitoring and operations tools  
Concurrency and recursion detection  
Asynchronous invocation  
Code signing  
File systems

### Execution role

Role name  
[lab-lambda-execution-role](#)

**Resource summary**

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs  
3 actions, 1 resource

By action | **By resource**

Resource	Actions
<b>All resources</b>	Allow: logs:CreateLogGroup Allow: logs:CreateLogStream Allow: logs:PutLogEvents

Lambda obtained this information from the following policy statements:

- Managed policy AWSLambdaBasicExecutionRole, statement 0

- In the IAM console, review the **permissions policies** attached to the role. These policies define what actions the function is allowed to perform on AWS services.
- Click on **Add permissions**, then select **Attach policies**.

IAM > Roles > lab-lambda-execution-role

Identity and Access Management (IAM)

Search IAM

Dashboard

▼ Access management

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings
- Root access management

▼ Access reports

- Access Analyzer
- External access
- Unused access
- Analyzer settings

### lab-lambda-execution-role

Summary

Creation date  
March 14, 2025, 14:43 (UTC-04:00)

Last activity  
-

ARN  
[arn:aws:iam::767397810700:role/lab-lambda-execution-role](#)

Maximum session duration  
1 hour

Permissions | Trust relationships | Tags (1) | Last Accessed | Revoke sessions

**Permissions policies (2)**

You can attach up to 10 managed policies.

Filter by Type  
All types

Policy name	Type	Attached entities
<input type="checkbox"/> <a href="#">AWSLambdaBasicExecutionRole</a>	AWS managed	1

**Add permissions**

- Attach policies
- Create inline policy

- Search for and select AmazonS3ReadOnlyAccess to grant the function read access to S3. This allows Lambda to list and read objects from S3 buckets but prevents it from modifying or deleting data.
- Click **Add permissions** to save changes.

## Attach policy to lab-lambda-execution-role

► Current permissions policies (1)

Other permissions policies (1/1037)

Search: AmazonS3ReadOnlyAccess Filter by Type: All types 1 match

<input checked="" type="checkbox"/>	Policy name	Type	Description
<input checked="" type="checkbox"/>	AmazonS3ReadOnlyAccess	AWS managed	Provides read only access to all buckets via ...

[Cancel](#) [Add permissions](#)

- Return to the Lambda console and refresh the **Execution Role** section to verify that S3 read permissions have now been added to the function.

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration | Triggers | **Permissions** | Destinations | Function URL | Environment variables | Tags | VPC | RDS databases | Monitoring and operations tools | Concurrency and recursion detection | Asynchronous invocation | Code signing | File systems

**Execution role** [Edit](#) [View role document](#)

Role name: lab-lambda-execution-role

**Resource summary**

To view the resources and actions that your function has permission to access, choose a service.

Amazon S3  
3 actions, 1 resource

By action | **By resource**

Resource	Actions
All resources	Allow: s3:Get* Allow: s3:List* Allow: s3:Describe*

Lambda obtained this information from the following policy statements:

- Managed policy AmazonS3ReadOnlyAccess, statement 0

With this change, the function now has explicit permission to interact with S3. However, for S3 to trigger Lambda, an additional **resource-based policy** must be in place, which is covered in the next section.


## Configure an S3 Event Trigger

To allow Amazon S3 to invoke the Lambda function automatically whenever a new object is created, an **S3 event trigger** must be added.

- Scroll up to the **Function overview** section.
- Click **Add trigger**, then select **S3** as the event source.
- Choose the S3 bucket lab-bucket-`<random-id>`. This bucket will send events to the function whenever an object is added to it.
- Keep **Event types** set to **All object create events**. This means that any new object uploaded to the bucket will trigger the Lambda function.
- Read the **Recursive Invocation** message, which warns that if the Lambda function writes data back to the same bucket, it may trigger itself in an infinite loop. Select the checkbox to acknowledge this behavior.
- Click **Add** to save the trigger configuration.

## Add trigger

Trigger configuration
[info](#)


S3
aws asynchronous storage

**Bucket**

Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

Bucket region: us-east-1

**Event types**

Select the events that you want to trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

☒ All object create events

**Prefix - optional**

Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters. Any [special characters](#) must be URL encoded.

**Suffix - optional**

Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters. Any [special characters](#) must be URL encoded.

**Recursive invocation**

If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)

☒ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel
Add

Now, whenever an object is created in lab-bucket-<random-id>, Amazon S3 will invoke the function.

## Review the Resource-Based Policy

Lambda automatically creates a **resource-based policy** when it is configured to be triggered by external services like Amazon S3. This policy defines which AWS services, accounts, or principals can invoke the function.

- Navigate to the **Configuration** tab, then open the **Permissions** section.
- Scroll down to **Resource-based policy statements** and locate the newly created policy. This policy allows the **Amazon S3** service to invoke the function.
- Select the policy and click **View policy** to examine the details.

**Permissions**

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

Monitoring and operations tools

Concurrency and recursion detection

Asynchronous invocation

Code signing

File systems

State machines

**lab-lambda-execution-role**

**Resource summary**

To view the resources and actions that your function has permission to access, choose a service.

Amazon CloudWatch Logs  
3 actions, 1 resource

By action | **By resource**

**All resources**

Allow: logs:CreateLogGroup  
Allow: logs:CreateLogStream  
Allow: logs:PutLogEvents

Lambda obtained this information from the following policy statements:

- Managed policy AWSLambdaBasicExecutionRole, statement 0

**Resource-based policy statements (1)**

Resource-based policies grant other AWS accounts and services permissions to access your Lambda resources.

Find policy statements

Statement ID	Principal	PrincipalOr...	Conditions	Action
lambda-4f1193c3-de01-401a-a...	s3.amazonaws.com	-	StringEquals, ArnLike	lambda:InvokeFunction

- The policy includes the following key attributes:

**Resource-based policy document**

```
1 {
2   "Version": "2012-10-17",
3   "Id": "default",
4   "Statement": [
5     {
6       "Sid": "lambda-4f1193c3-de01-401a-ad7d-c350275b47d3",
7       "Effect": "Allow",
8       "Principal": {
9         "Service": "s3.amazonaws.com"
10      },
11      "Action": "lambda:InvokeFunction",
12      "Resource": "arn:aws:lambda:us-east-1:767397810700:function:eventLogger",
13      "Condition": {
14        "StringEquals": {
15          "AWS:SourceAccount": "767397810700"
16        },
17        "ArnLike": {
18          "AWS:SourceArn": "arn:aws:s3:::lab-bucket-767397810700-38e2f3e0"
19        }
20      }
21    }
22  ]
23 }
```

1:1 JSON Spaces: 2

Close

- It allows the `lambda:InvokeFunction` action, which grants Amazon S3 permission to trigger the Lambda function.
- The **Service Principal** is set to `s3.amazonaws.com`, which means that only the Amazon S3 service is authorized to invoke this function.
- The **Resource ARN** references the specific S3 bucket `lab-bucket-767397810700-38e2f3e0` in this AWS account, ensuring that only this bucket can send events to the function.
- With this configuration, the function now has the necessary **IAM execution role permissions** to read from S3 and a **resource-based policy** that allows S3 to invoke it securely.

- Close the policy popup.

#### Task 4 - Analyze the Event and Context Objects

The **event object** carries input data to a Lambda function, providing details about the triggering event. For example, if an S3 event triggers the function, the event object contains metadata about the bucket and object that initiated the invocation.

The **context object** provides runtime metadata about the function's execution environment, including memory allocation, remaining execution time, request ID, and logging details. Additionally, the **log output** also captures environment variables configured for the function, helping you verify that they are correctly set and accessible during execution.

This task will guide you through inspecting and using these objects to better understand how Lambda functions process incoming data and interact with their execution environment.

#### Create, Inspect and Execute a Test Event

- Click on the **Test** tab in the Lambda function console.
- Click **Create new test event** to define a test case that simulates an event triggering the function.
- Name the event **S3TestEvent** for easy identification.
- Select **S3 Put** as the event template. This template represents an S3 **PutObject** event, which occurs when a file is uploaded to an S3 bucket.
- Click **Save** to save the test event.

The screenshot shows the AWS Lambda console interface for creating a test event. The breadcrumb navigation at the top indicates the path: Lambda > Functions > eventLogger. The 'Test' tab is selected in the top navigation bar. The 'Test event' section has a title bar with 'Info' and buttons for 'CloudWatch Logs Live Tail', 'Save', and 'Test'. Below this, a message states: 'To invoke your function without saving an event, configure the JSON event, then choose Test.' The 'Test event action' dropdown is set to 'Create new event'. The 'Event name' field contains 'S3TestEvent'. The 'Event sharing settings' section shows 'Private' selected. The 'Template - optional' dropdown is set to 's3-put'. The 'Event JSON' section is visible at the bottom.

- Observe the event structure, especially the Records array, which contains metadata about the event. Notice the following key details:
  - Under s3 > bucket, you will find the name of the bucket where the object was uploaded.
  - Under s3 > object, you will see details about the uploaded file, such as its key (file name) and size.
  - By examining these properties, you can understand how your Lambda function receives and processes event data from S3.



Event JSON

```
9 *   "userIdentity": {
10 *     "principalId": "EXAMPLE"
11 *   },
12 *   "requestParameters": {
13 *     "sourceIPAddress": "127.0.0.1"
14 *   },
15 *   "responseElements": {
16 *     "x-amz-request-id": "EXAMPLE123456789",
17 *     "x-amz-id-2": "EXAMPLE123/5678abcdefgijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
18 *   },
19 *   "s3": {
20 *     "s3SchemaVersion": "1.0",
21 *     "configurationId": "testConfigRule",
22 *     "bucket": {
23 *       "name": "example-bucket",
24 *       "ownerIdentity": {
25 *         "principalId": "EXAMPLE"
26 *       },
27 *       "arn": "arn:aws:s3:::example-bucket"
28 *     },
29 *     "object": {
30 *       "key": "test%2Fkey",
31 *       "size": 1024,
32 *       "eTag": "0123456789abcdef0123456789abcdef",
33 *       "sequencer": "0A1B2C3D4E5F678901"
34 *     }
35 *   }
36 * }
37 * ]
38 * }
```

Format JSON

- Click **Test** to invoke the function using the **S3TestEvent** data.

Lambda > Functions > eventLogger

Code **Test** Monitor Configuration Aliases Versions

Executing function: succeeded (logs 2)

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "event": {
    "Records": [
      {
        "eventVersion": "2.0",
        "eventSource": "aws:s3",
        "awsRegion": "us-east-1",
        "eventTime": "1970-01-01T00:00:00.000Z",
        "eventName": "ObjectCreated:Put",
        "userIdentity": {
          "principalId": "EXAMPLE"
        }
      }
    ]
  }
}
```

Summary

<b>Code SHA-256</b> YcqZJFC3a7HGCxpGRnS1IDzszXczy5wZM+me3L9J3SY=	<b>Execution time</b> 21 seconds ago
<b>Request ID</b> f2028671-dc4e-473b-ac43-20d6c59d44f0	<b>Function version</b> \$LATEST
<b>Init duration</b> 136.80 ms	<b>Duration</b> 7.92 ms
<b>Billed duration</b> 8 ms	<b>Resources configured</b> 512 MB
<b>Max memory used</b> 73 MB	

- After execution, review the **log output** in the Lambda console to examine how the function processed the event. The log output will include:
  - Event data:** The full JSON payload received by the function, representing the triggering event.
  - Context data:** Metadata about the execution environment, including the function's request ID and memory usage.
  - Environment variables:** The values of all environment variables set for the function, such as APP\_NAME and APP\_VERSION. This confirms that the function is correctly retrieving configuration values at runtime.
  - Remaining execution time:** The output of context.getRemainingTimeInMillis(), which indicates how much time is left before the function reaches its timeout. This can be useful for optimizing performance and handling time-sensitive operations.

73 MB

### Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```

START RequestId: f2028671-dc4e-473b-ac43-20d6c59d44f0 Version: $LATEST
2025-03-14T19:42:49.062Z      f2028671-dc4e-473b-ac43-20d6c59d44f0  INFO  Remaining time in milliseconds: 59998
2025-03-14T19:42:49.065Z      f2028671-dc4e-473b-ac43-20d6c59d44f0  INFO  App Name: lab-app
2025-03-14T19:42:49.065Z      f2028671-dc4e-473b-ac43-20d6c59d44f0  INFO  App Version: v1
2025-03-14T19:42:49.066Z      f2028671-dc4e-473b-ac43-20d6c59d44f0  INFO  Event: {
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",

```

Test event [Info](#) [CloudWatch Logs Live Tail](#) [Save](#) [Test](#)

---

73 MB

### Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```

    },
    "arn": "arn:aws:s3:::example-bucket"
  },
  "object": {
    "key": "test%2Fkey",
    "size": 1024,
    "eTag": "0123456789abcdef0123456789abcdef",
    "sequencer": "0A1B2C3D4E5F678901"
  }
}
}
2025-03-14T19:42:49.066Z      f2028671-dc4e-473b-ac43-20d6c59d44f0  INFO  Context: {
  "callbackWaitsForEmptyEventLoop": true,
  "functionVersion": "$LATEST",
  "functionName": "eventLogger",
  "memoryLimitInMB": "512",
  "logGroupName": "/aws/lambda/eventLogger",
  "logStreamName": "2025/03/14/[$LATEST]d967ea9a8ee4ffe8b4788812a02d277",
  "invokedFunctionArn": "arn:aws:lambda:us-east-1:767397810700:function:eventLogger",
  "awsRequestId": "f2028671-dc4e-473b-ac43-20d6c59d44f0"
}
2025-03-14T19:42:49.066Z      f2028671-dc4e-473b-ac43-20d6c59d44f0  INFO  Remaining time in milliseconds: 59994
END RequestId: f2028671-dc4e-473b-ac43-20d6c59d44f0
REPORT RequestId: f2028671-dc4e-473b-ac43-20d6c59d44f0 Duration: 7.92 ms Billed Duration: 8 ms Memory Size: 512 MB Max Memory Used: 73 MB Init Duration: 136.80 ms

```

By analyzing the **event**, **context**, and **environment variables** in the log output, you gain deeper insights into how Lambda functions interact with AWS services, manage execution constraints, and utilize configuration settings dynamically.

## Task 5 - Test the Function

Testing the Lambda function with a real event helps verify its behavior and ensures it correctly processes incoming data. In this task, you will upload a file to an **S3 bucket**, triggering the function, and then analyze the logs in **Amazon CloudWatch** to inspect the event data, context metadata, and environment variables captured during execution.

Trigger the Function with an S3 Event

- Upload a [test file](#) to the **S3 bucket** lab-bucket-<random-id>.

Amazon S3 > Buckets > lab-bucket-767397810700-38e2f3e0

### lab-bucket-767397810700-38e2f3e0 [Info](#)

[Objects](#) [Metadata](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (1) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

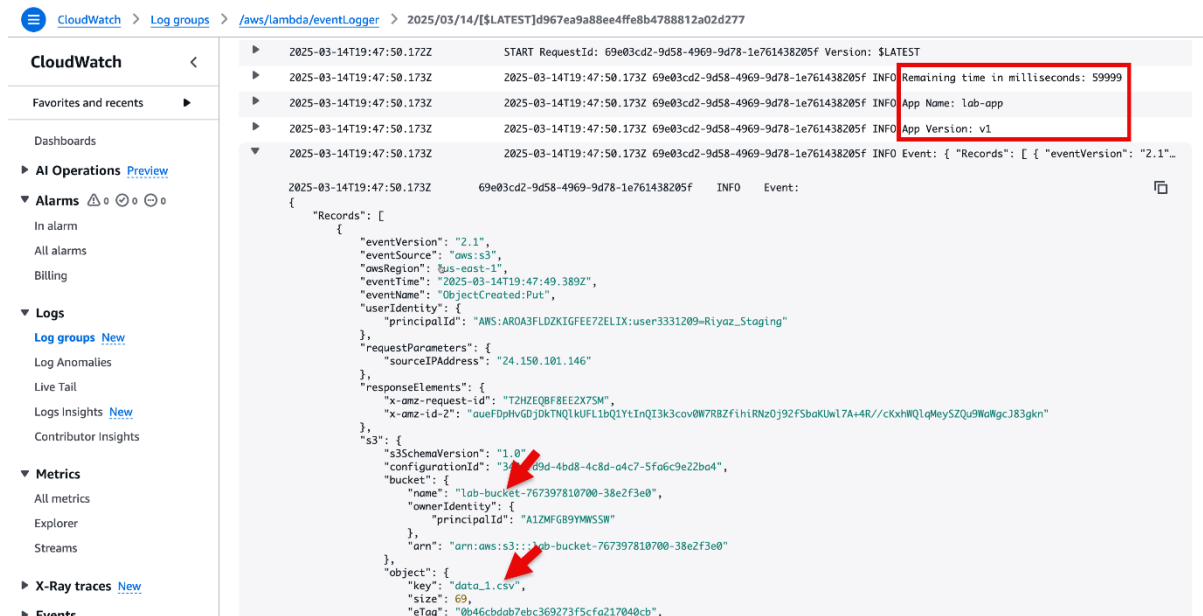
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	<a href="#">data_1.csv</a>	csv	March 14, 2025, 15:47:50 (UTC-04:00)	69.0 B	Standard

- This action will generate an **S3 PutObject event**, which will automatically trigger the Lambda function.

Since the function is already configured to process S3 events, it will receive the event data containing details about the uploaded file, including the bucket name, object key (file name), and other metadata.

## Review Logs in CloudWatch

- From the **Monitor** tab of your Lambda function, click **View CloudWatch logs**.
- Click on the most recent log stream to view the execution logs for the triggered invocation.



CloudWatch > Log groups > /aws/lambda/eventLogger > 2025/03/14/[LATEST]d967ea9a88ee4ffe8b4788812a02d277

CloudWatch

2025-03-14T19:47:50.172Z START RequestId: 69e03cd2-9d58-4969-9d78-1e761438205f Version: \$LATEST

2025-03-14T19:47:50.173Z 2025-03-14T19:47:50.173Z 69e03cd2-9d58-4969-9d78-1e761438205f INFO Remaining time in milliseconds: 59999

2025-03-14T19:47:50.173Z 2025-03-14T19:47:50.173Z 69e03cd2-9d58-4969-9d78-1e761438205f INFO App Name: lab-app

2025-03-14T19:47:50.173Z 2025-03-14T19:47:50.173Z 69e03cd2-9d58-4969-9d78-1e761438205f INFO App Version: v1

2025-03-14T19:47:50.173Z 2025-03-14T19:47:50.173Z 69e03cd2-9d58-4969-9d78-1e761438205f INFO Event: { "Records": [ { "eventVersion": "2.1"...

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "2025-03-14T19:47:49.389Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:ARO3FLDZKIGFEE7ZELIX:user3331209-Riyaz_Staging"
      },
      "requestParameters": {
        "sourceIPAddress": "24.150.101.146"
      },
      "responseElements": {
        "x-amz-request-id": "TZHZEQBF8EE2X75M",
        "x-amz-id-2": "auefDphv6DjDkTnQ1kuFL1bQ1YtInQ13k3cov0W7R8ZfihRNz0j9Zf5baKUw17A+4R//cKxhW1qMeySZQu9WalfgcJ83gkn"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "33a69d-4bd8-4c8d-a4c7-5fa6c9e22b04",
        "bucket": {
          "name": "lab-bucket-767397810700-38e2f3e0",
          "ownerIdentity": {
            "principalId": "A1ZMFGB9YMWSSW"
          },
          "arn": "arn:aws:s3:::lab-bucket-767397810700-38e2f3e0"
        },
        "object": {
          "key": "data_1.csv",
          "size": 69,
          "eTag": "0b46cbdb7ebc369273f5cfa217040cb",

```

- The log output will include:
  - Event data:** The JSON payload received by the function, containing details about the uploaded S3 object.
  - Context data:** Metadata about the execution, including request ID, memory usage, and invocation duration.
  - Environment variables:** Values of predefined variables such as APP\_NAME and APP\_VERSION, confirming they are accessible at runtime.
  - Remaining execution time:** The output of context.getRemainingTimeInMillis(), which indicates how much time was left before the function completed execution.

By reviewing the CloudWatch logs, you can verify that the function was triggered successfully and processed the S3 event correctly. This also allows you to troubleshoot any issues by inspecting the exact event data received by the function.

## Task 6 - Clean up

To avoid unnecessary charges, you should delete resources after completing the lab.

- Navigate to the **AWS Lambda** console.
- Select the eventLogger function.
- Click **Actions** and choose **Delete function** to delete it if no longer needed.
- In the **IAM** console, locate the execution role and delete it if no longer needed.
- Empty the S3 bucket lab-bucket-<random-id> and delete it.

- In **CloudWatch Logs**, delete the log group associated with the function to free up storage.
- Go to the **CloudFormation** console and delete the lab-stack-10-2 stack to remove all lab infrastructure.