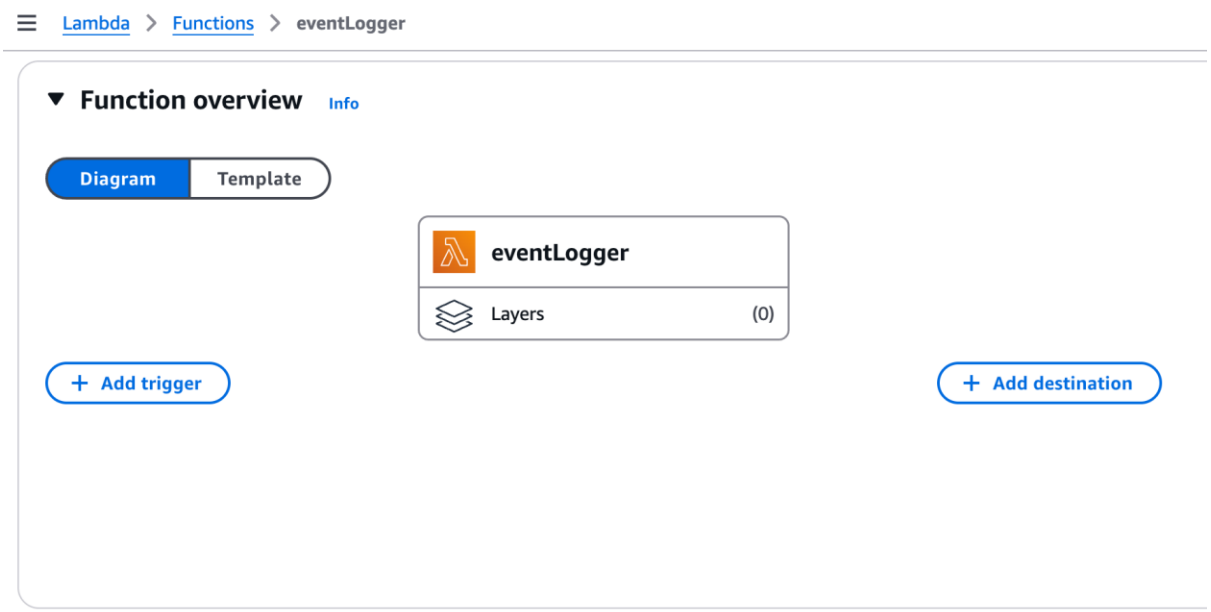
AWS-Lambda- Assignment1
SHASHANK BG

Lab tasks

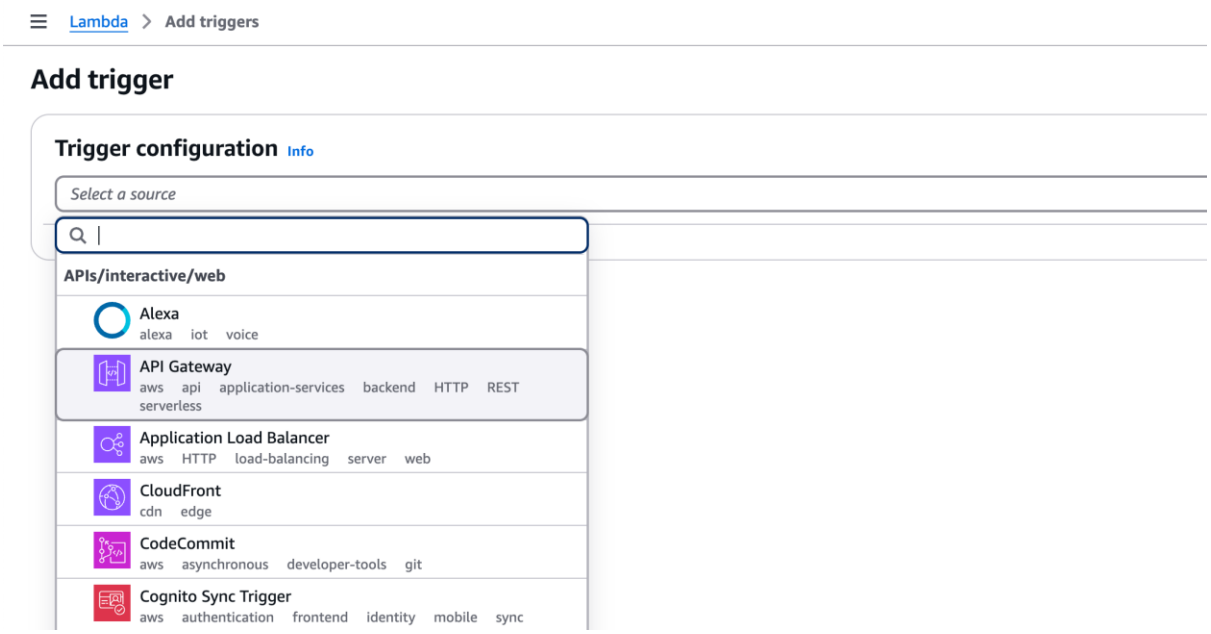
Task 1 - Navigate the AWS Lambda Console

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.

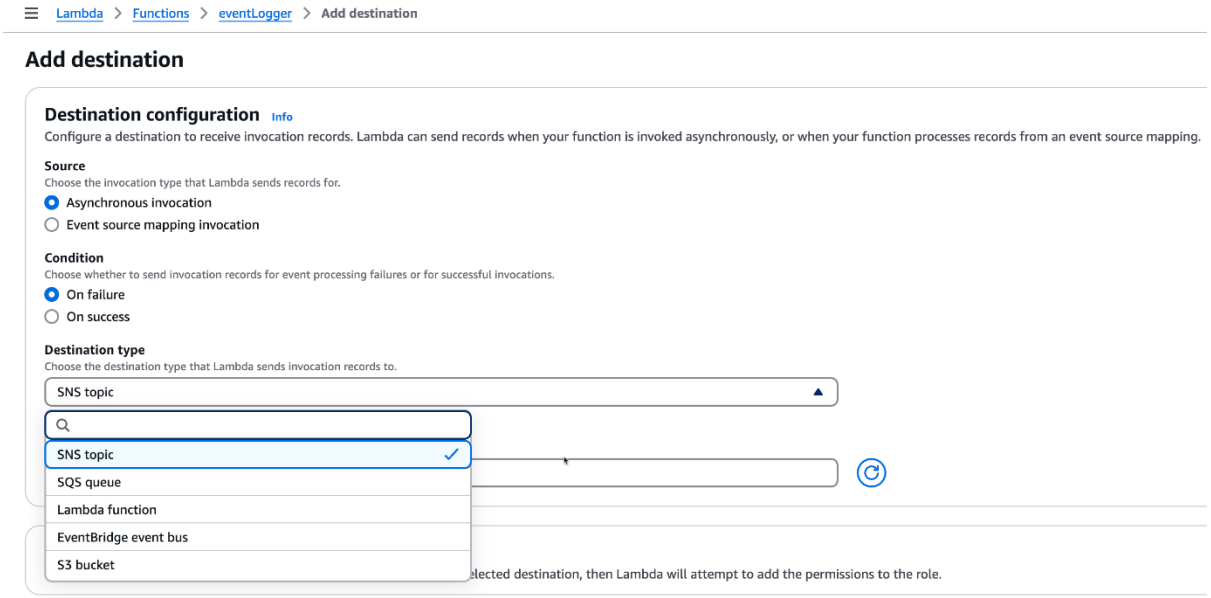
- Open the **AWS Management Console** and navigate to **AWS Lambda**.
- From the **Functions** section, open the function eventLogger.
- In the **Function overview** section, you'll see the Diagram section that displays the Layers at the center, and source event triggers on the left, and destinations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



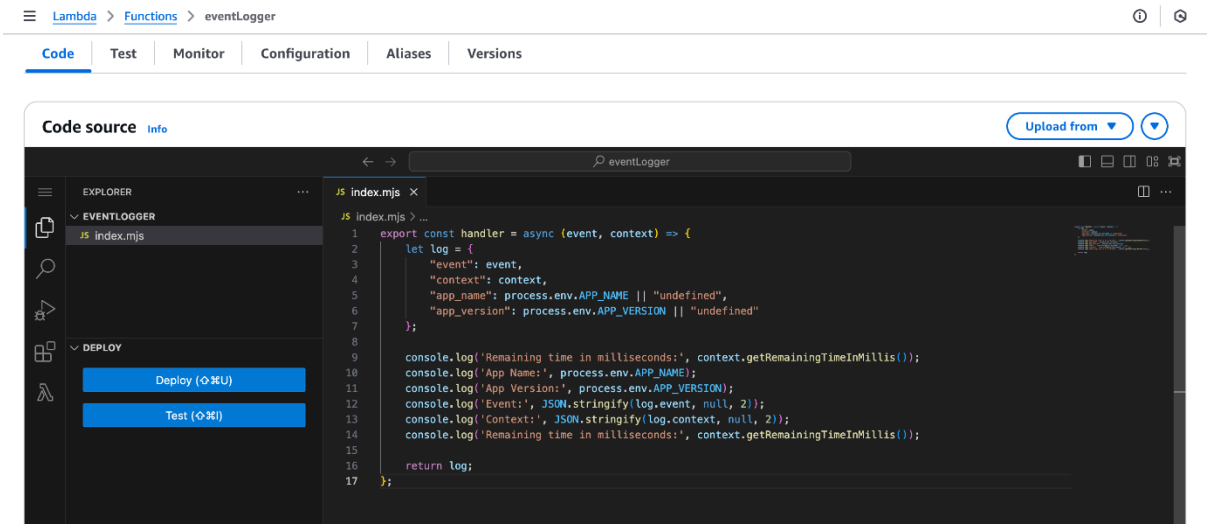
- Click on **Add trigger** to explore different ways Lambda functions can be invoked. Here you can choose and configure one or more triggers for your function. Triggers can be synchronous, like **API Gateway** and **ALB**, allowing real-time execution, or asynchronous, like **S3** and **SNS**, where 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.



- Click **Cancel** to go back.
- Click on **Add destination** to explore available options. Here you define what happens after the function executes. Destinations allow Lambda to send results of asynchronous invocations to services like **SQS, SNS, EventBridge, S3 bucket**, or another **Lambda function**. Success destinations handle successful execution, while failure destinations capture errors for debugging and monitoring, improving reliability in serverless applications.



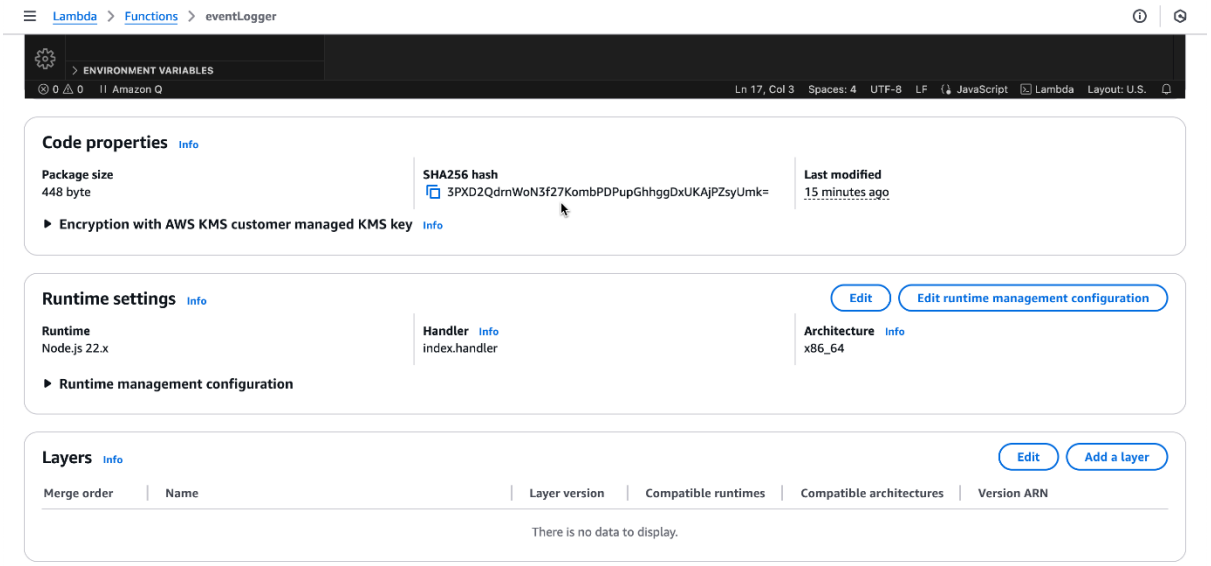
- Click **Cancel** to go back.
- Explore the **Code** tab.



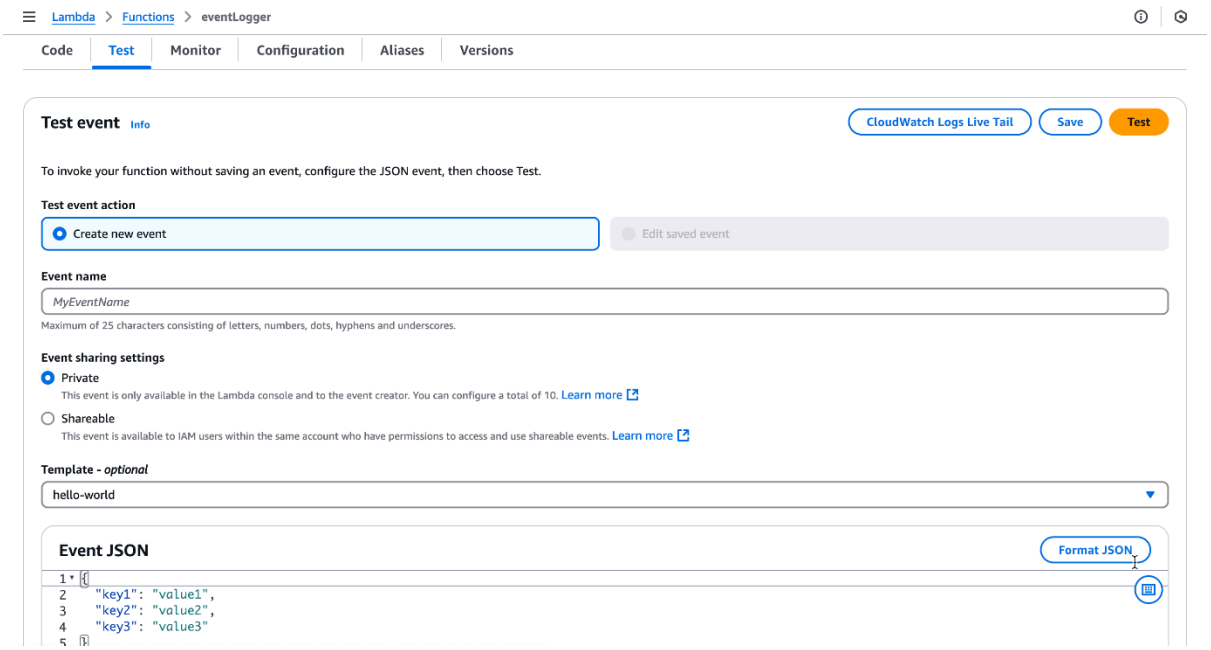
- The **Code source** panel provides you with a web-based IDE to write, test and deploy your code. This also provides option to use CloudWatch Live Tail feature.

```
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. };
```

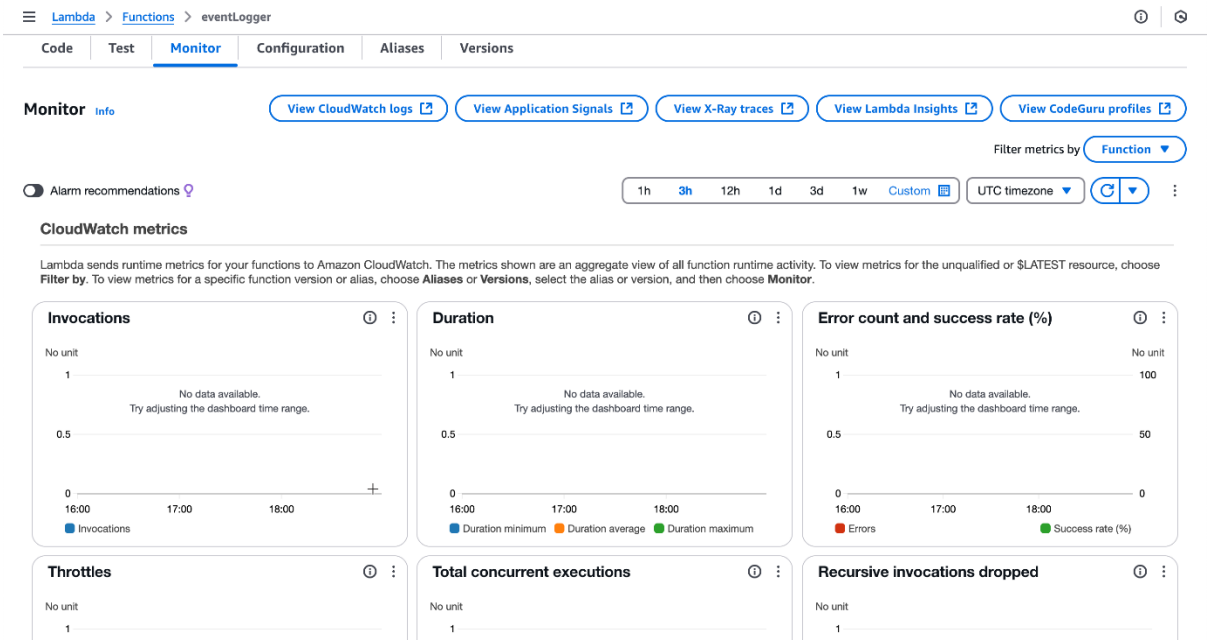
- Scroll down to review **Code properties**, **Runtime settings** and **Layers**. Explore and review different configuration options available.



- 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.

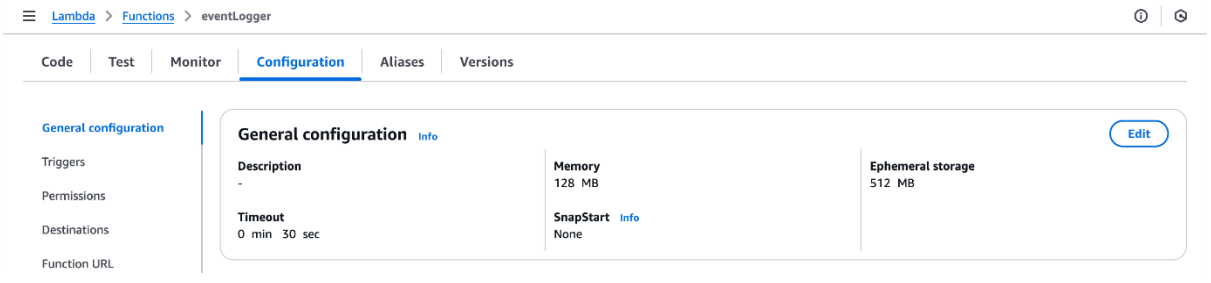


- Review the **Monitor** tab. This section provides essential monitoring and logging tools:



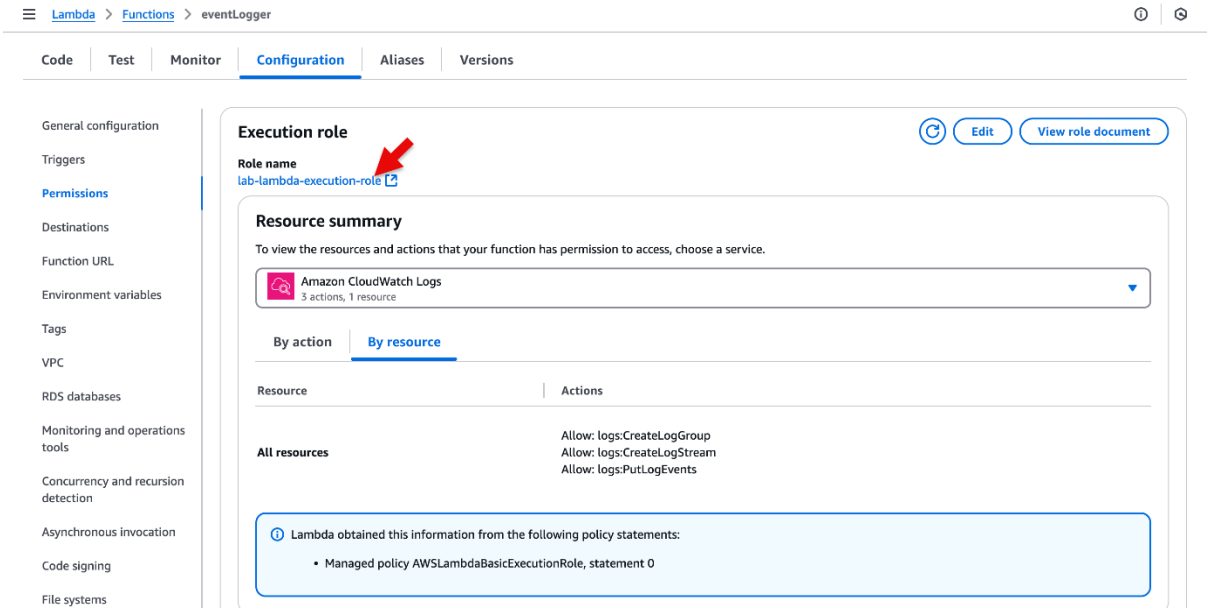
- **CloudWatch Logs**: View detailed execution logs, errors, and debugging information.
- **X-Ray Traces**: Analyze request flow, latency, and function dependencies.
- **Lambda Insights**: Monitor function execution metrics like duration, memory usage, error rates, and invocation counts.
- These monitoring 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.

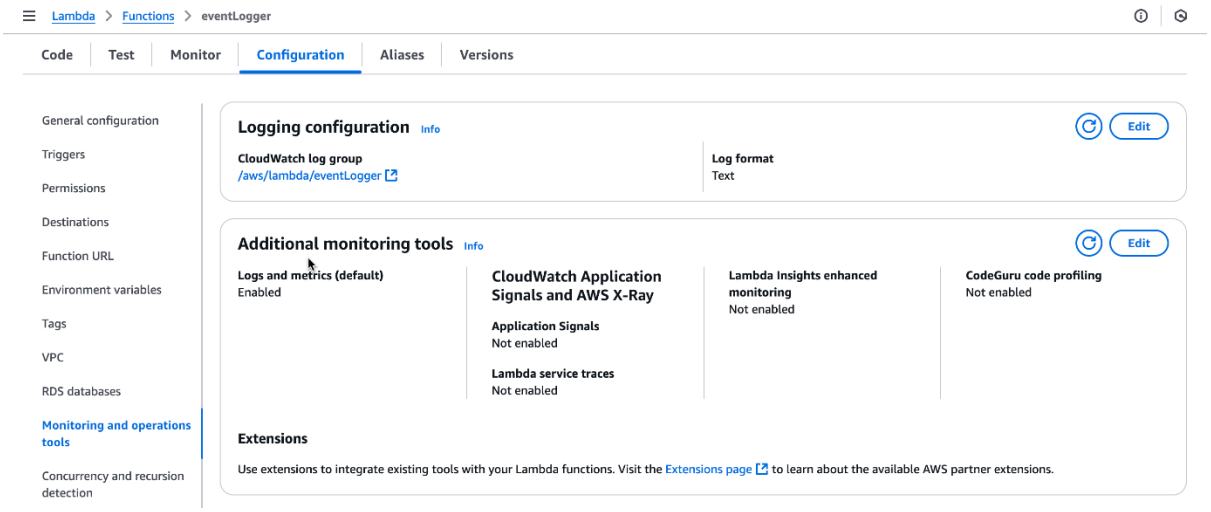


- **Triggers**: Here, you can add and manage event sources that invoke the function. This includes configuring event-driven invocations using services such as API Gateway, S3, DynamoDB Streams, and SQS.

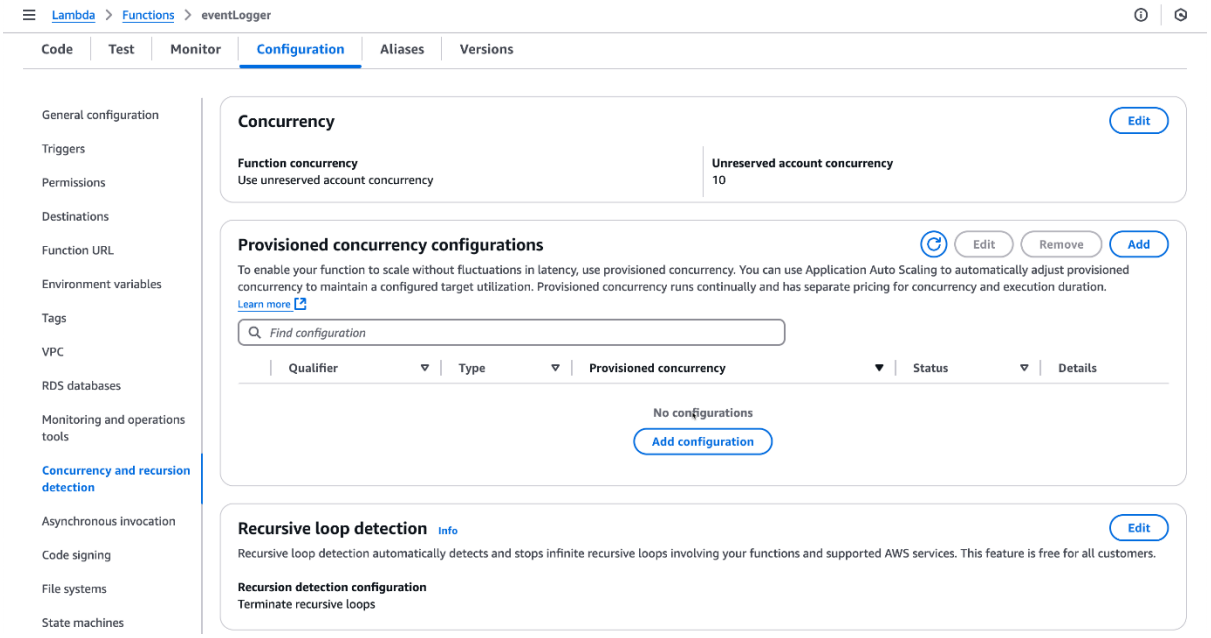
- **Permissions**: This section displays IAM permissions assigned to the function. The **lab-lambda-execution-role** includes AWSLambdaBasicExecutionRole policy providing the function CloudWatch logging access.



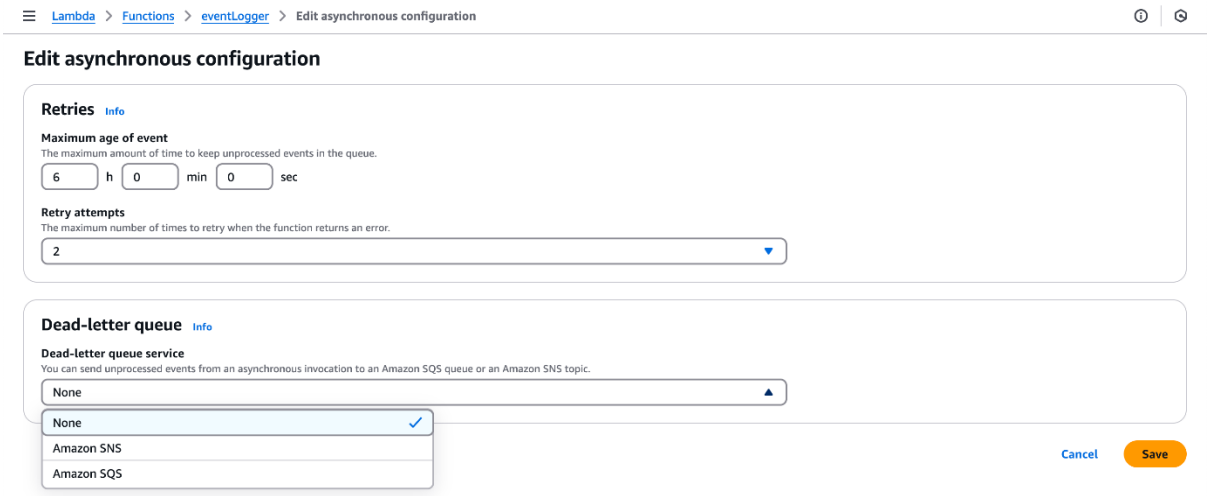
- **Destinations**: This section allows you to specify where execution results should be sent, such as SQS, SNS, another Lambda function, or an S3 bucket.
- **Function URL**: Here, you can set up a dedicated HTTPS endpoint to invoke the function directly without requiring an API Gateway integration.
- **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.



- **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.



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



- **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.

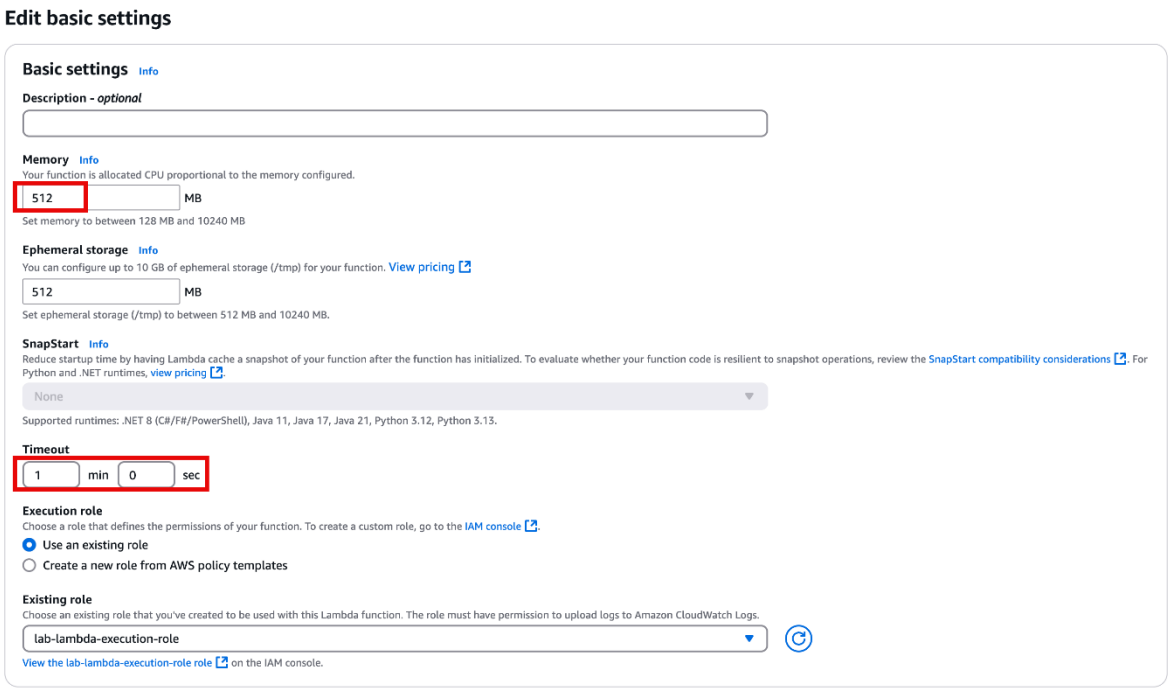
Task 2 - Modify Memory, Timeout, Environment Variables, and VPC Settings

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.

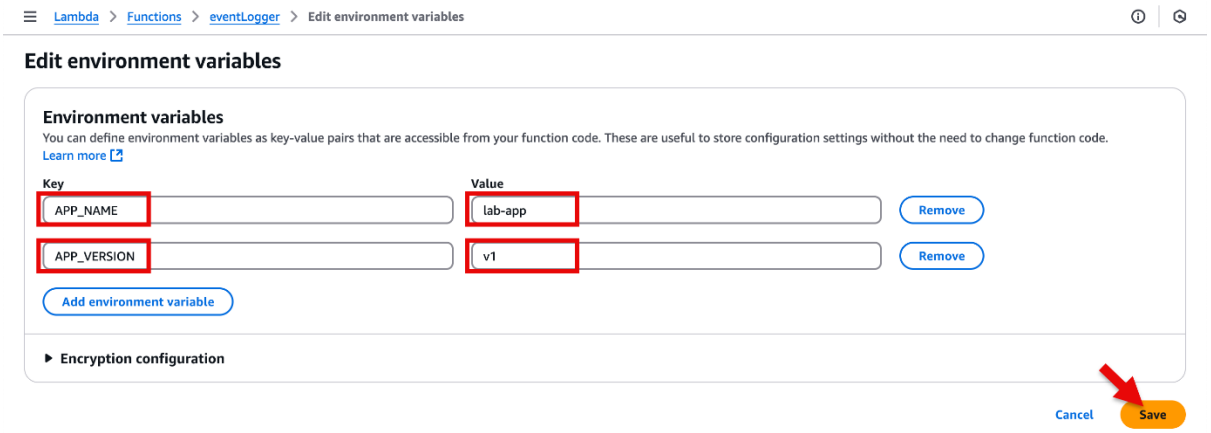


- Click **Save** to apply these changes.

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.



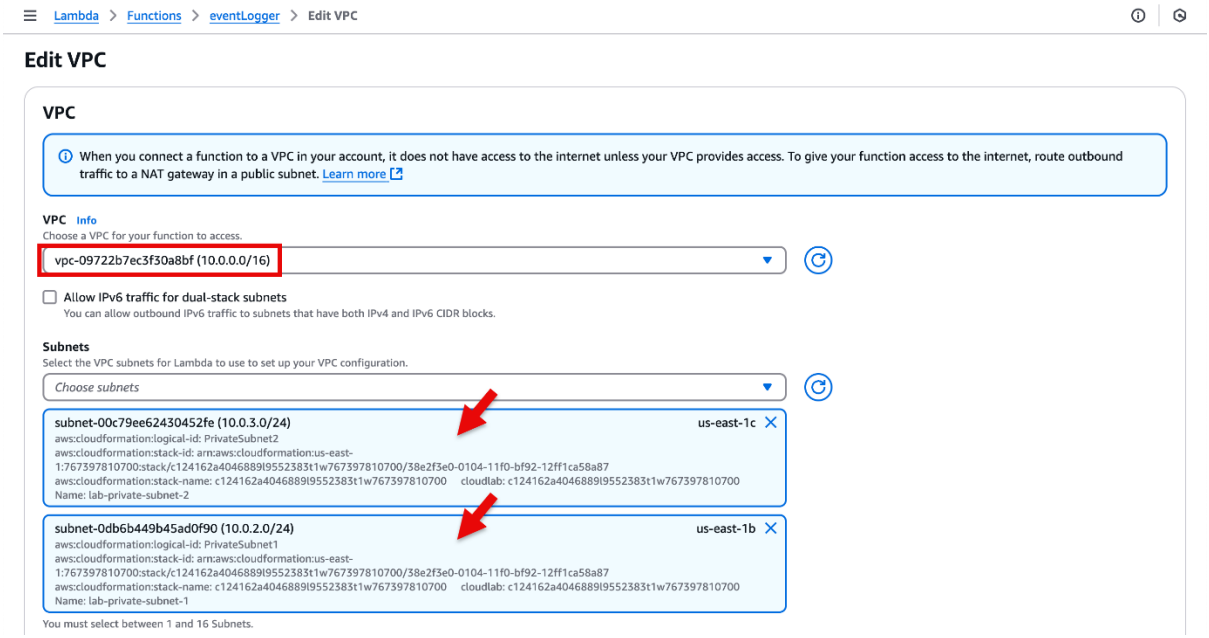
Configure VPC Settings

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**.

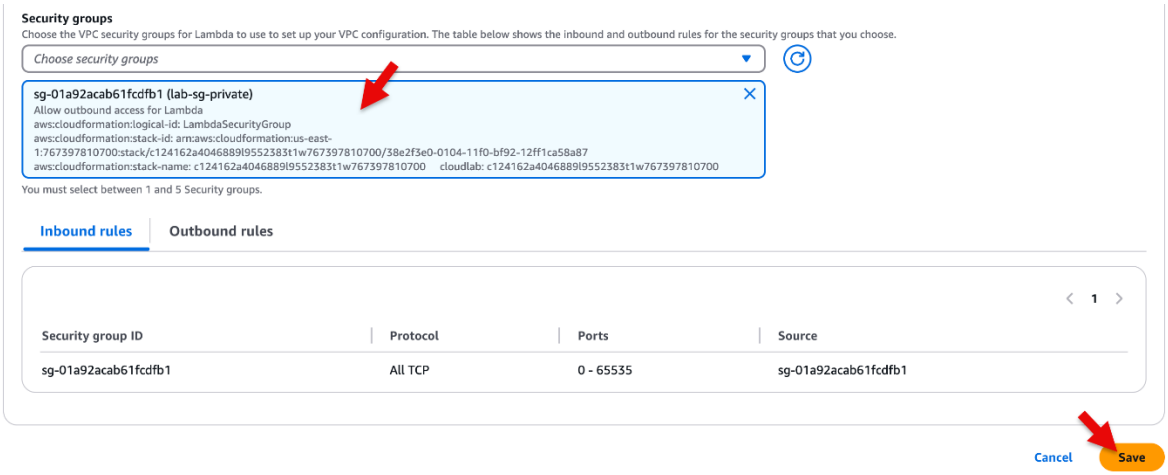
Note that this configuration is only needed for accessing resources that reside in a VPC. If you do not need to access any VPC resources, this configuration is not required.

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.

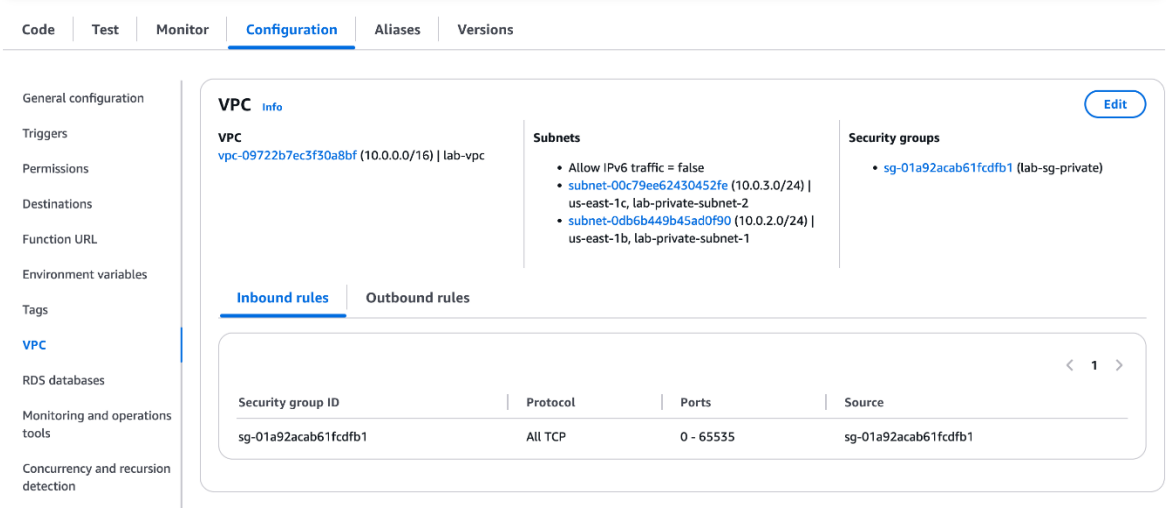
- Scroll to **VPC** section and click **Edit**.
- From the **VPC dropdown**, select lab-vpc. This allows the function to operate within the private network.
- Choose **two private subnets** from lab-vpc. Private subnets ensure the function remains secure by preventing direct internet access while allowing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



- 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.



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



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

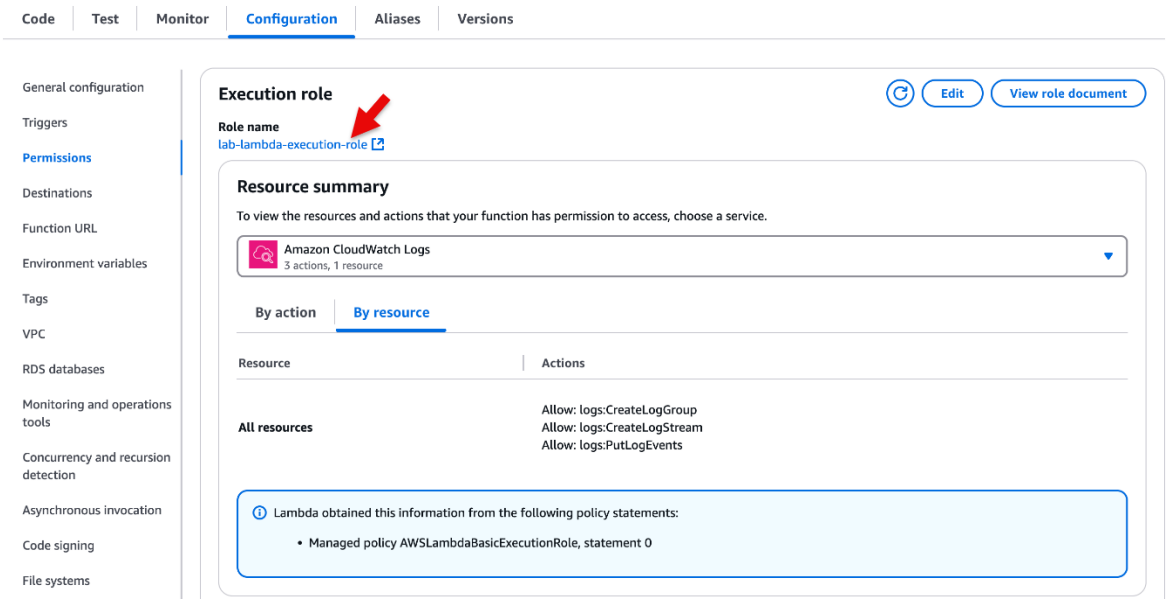
AWS Lambda uses **IAM roles** and **resource-based policies** to securely control access to AWS services and external resources. This task will help you understand how Lambda interacts with AWS services using execution roles and function policies.

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.

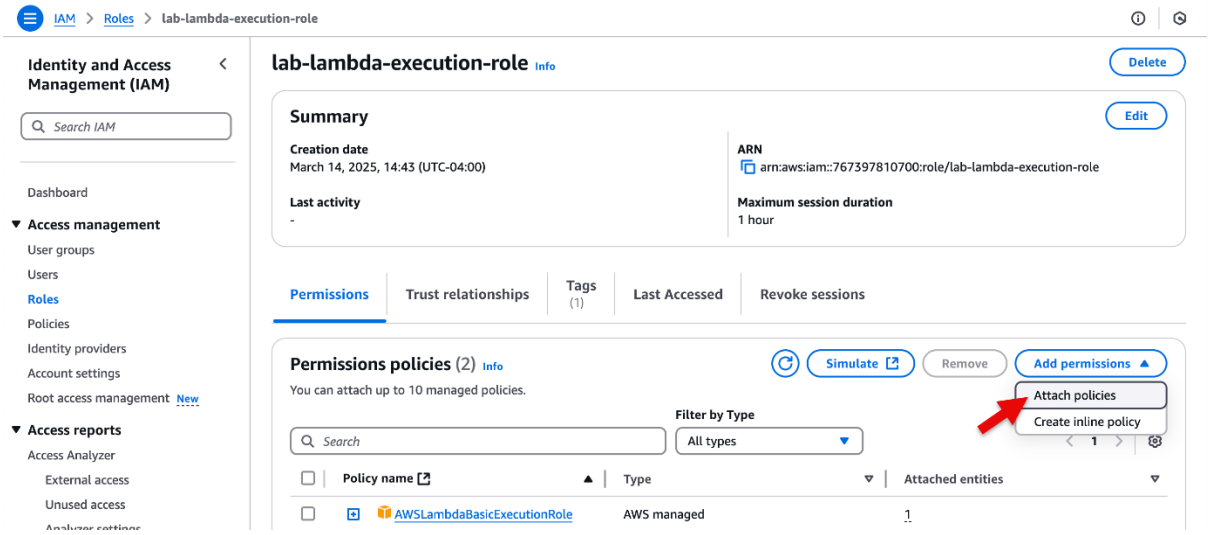
Review Execution Role and Add Permissions

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.

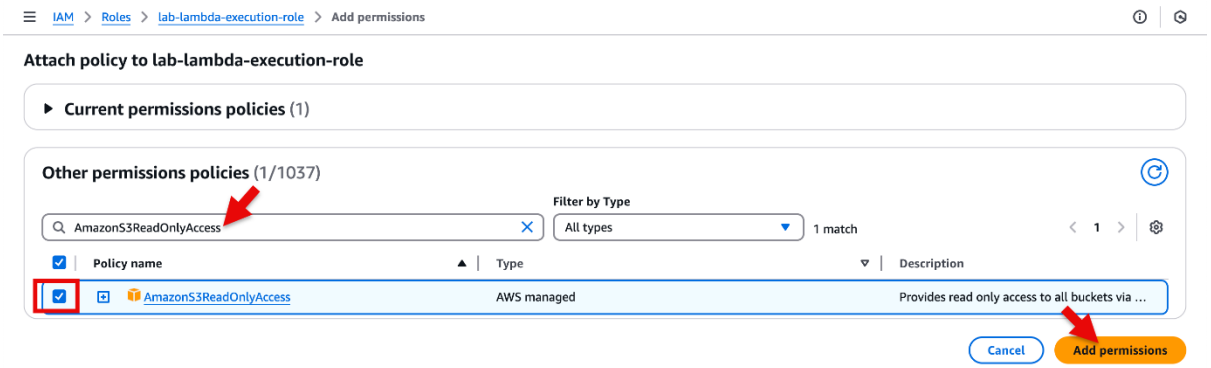
- In the **Permissions** section of the Lambda configuration, locate **Execution role** and click on the role name. This will redirect you to the IAM console.



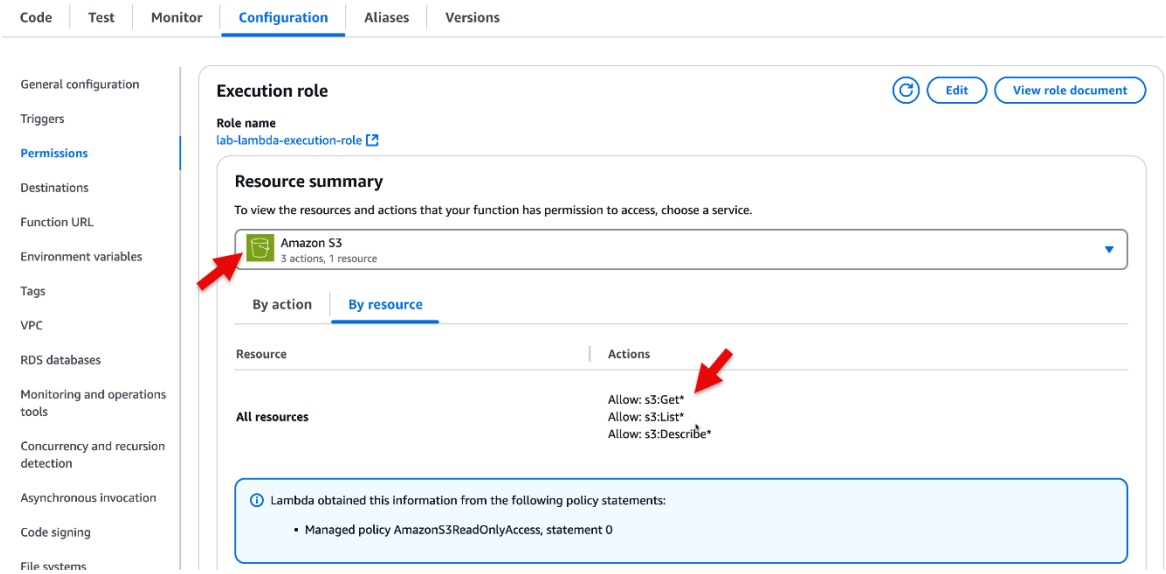
- In the IAM console, review the **permissions policies** attached to the role. These policies define what actions the function is allowed to perform on AWS services.
- Click on **Add permissions**, then select **Attach policies**.



- 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.



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

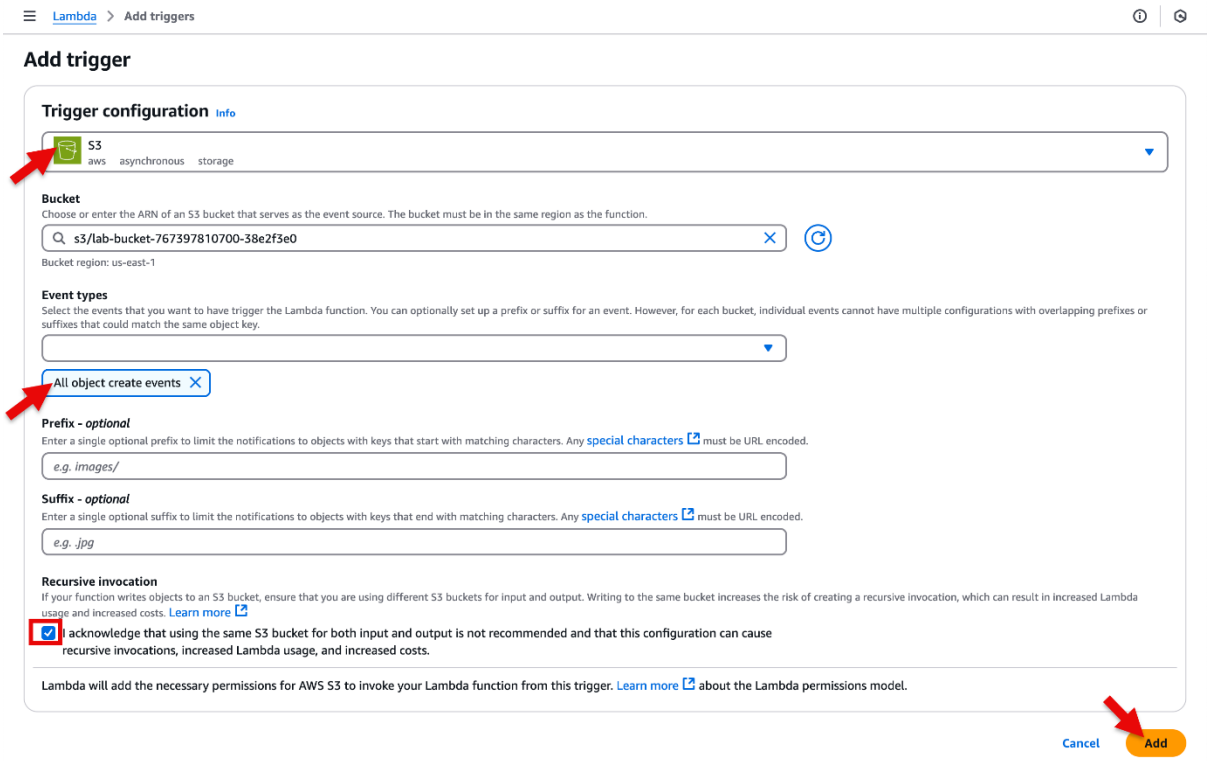


With this change, the function now has explicit permission to interact with S3. However, for S3 to trigger Lambda, an additional **resource-based policy** must be in place, which is covered in the next section.

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.

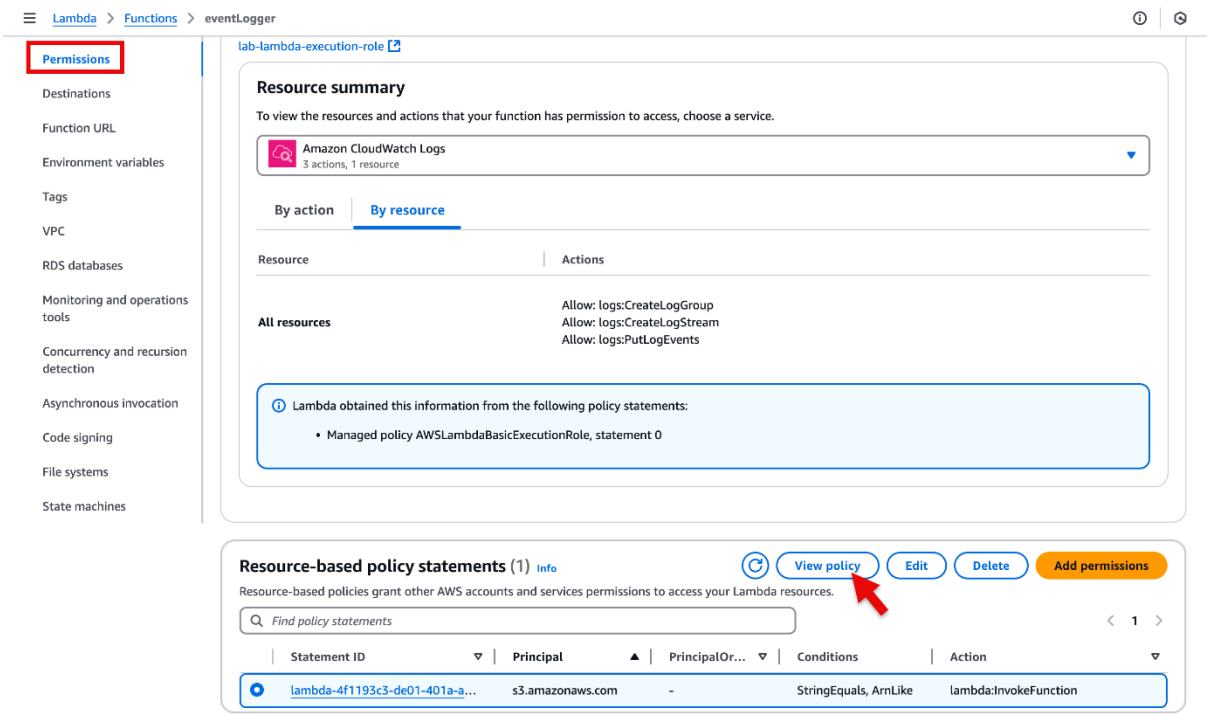


Now, whenever an object is created in lab-bucket-<random-id>, Amazon S3 will invoke the function.

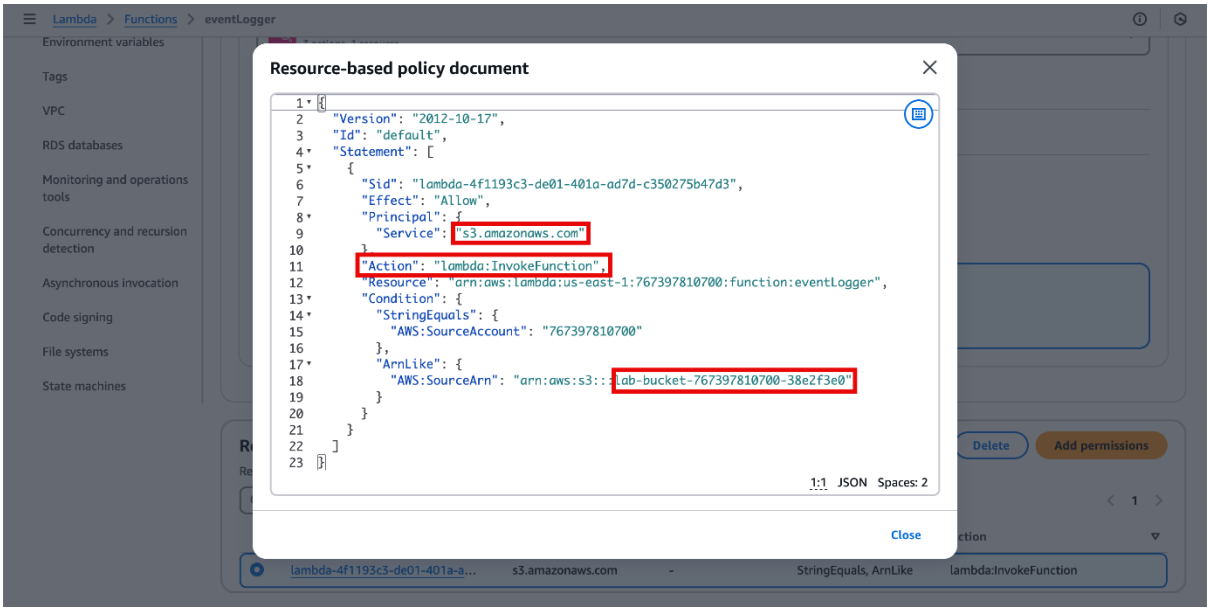
Review the Resource-Based Policy

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.



- The policy includes the following key attributes:



- 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-<random-id> 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

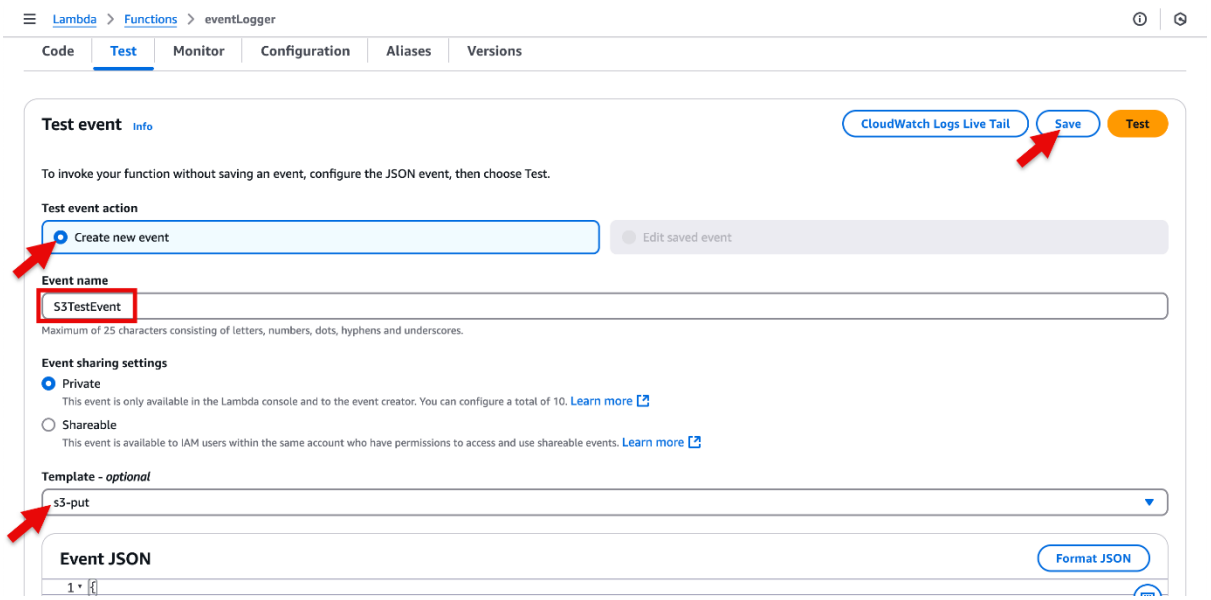
The **event object** carries input data to a Lambda function, providing details about the triggering event. For example, if an S3 event triggers the function, the event object contains metadata about the bucket and object that initiated the invocation.

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.

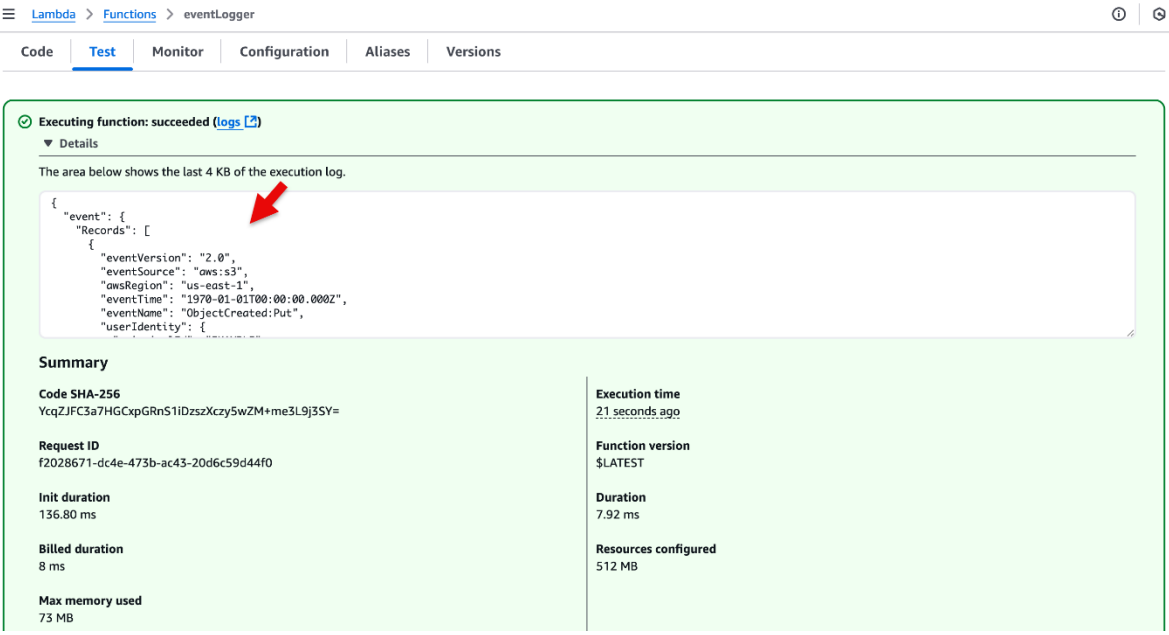
Create, Inspect and Execute a Test Event

- 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.

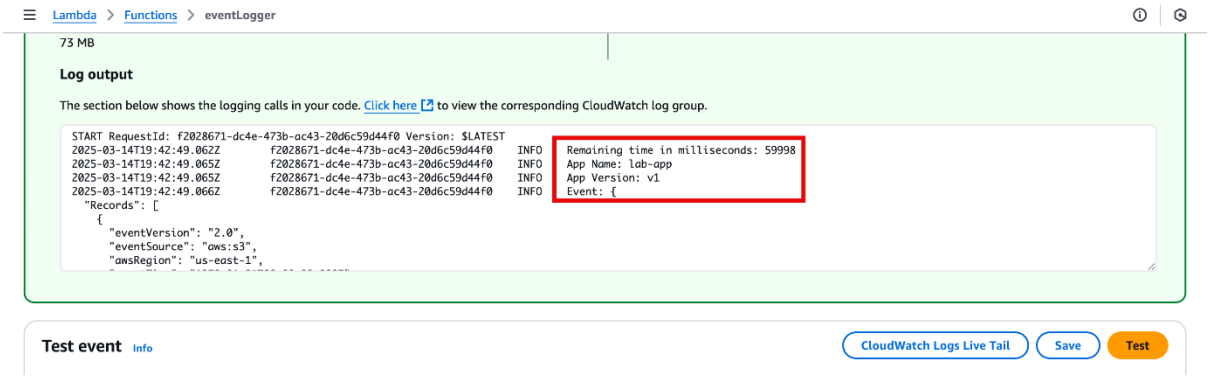


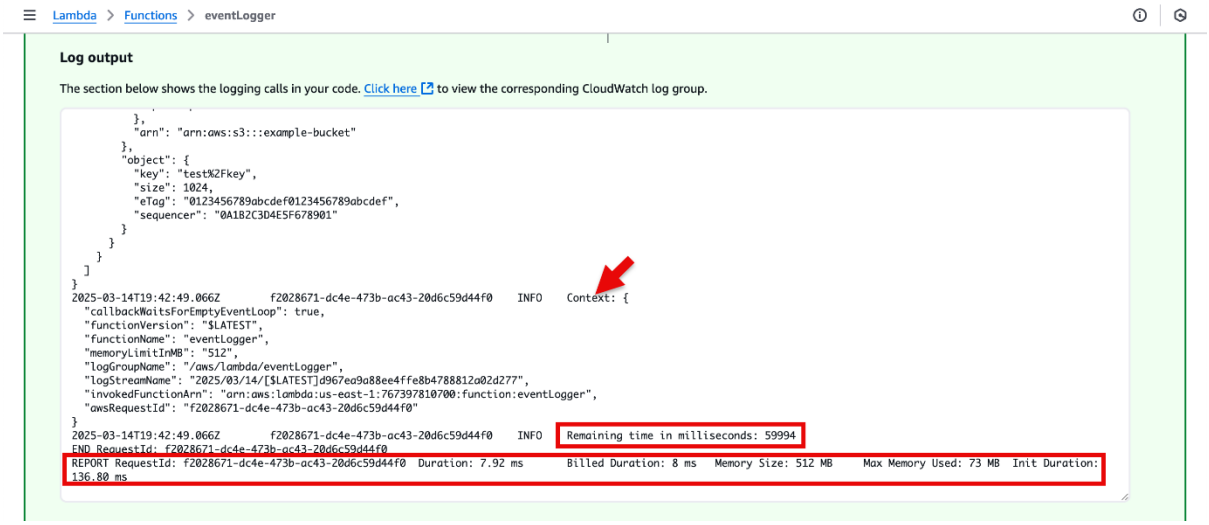
- Observe the event structure, especially the Records array, which contains metadata about the event. Notice the following key details:
    - Under s3 > bucket, you will find the name of the bucket where the object was uploaded.
    - Under s3 > object, you will see details about the uploaded file, such 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.

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



- 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.





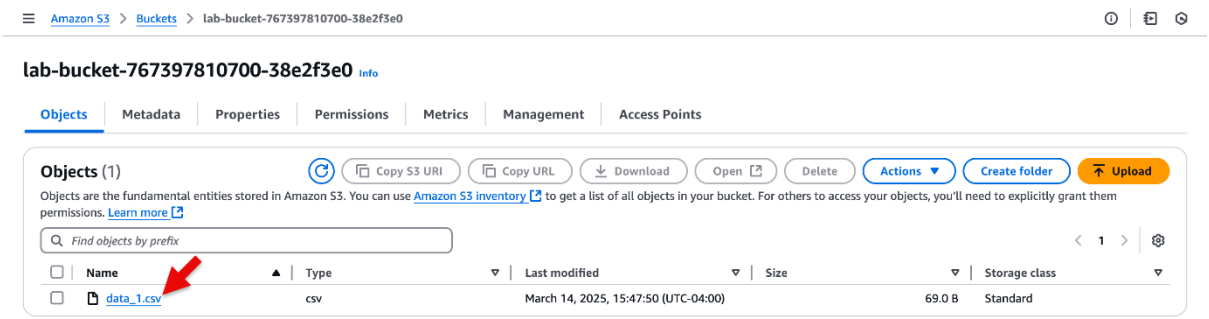
By analyzing the **event**, **context**, and **environment variables** in the log output, you gain deeper insights into how Lambda functions interact with AWS services, manage execution constraints, and utilize configuration settings dynamically.

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>.

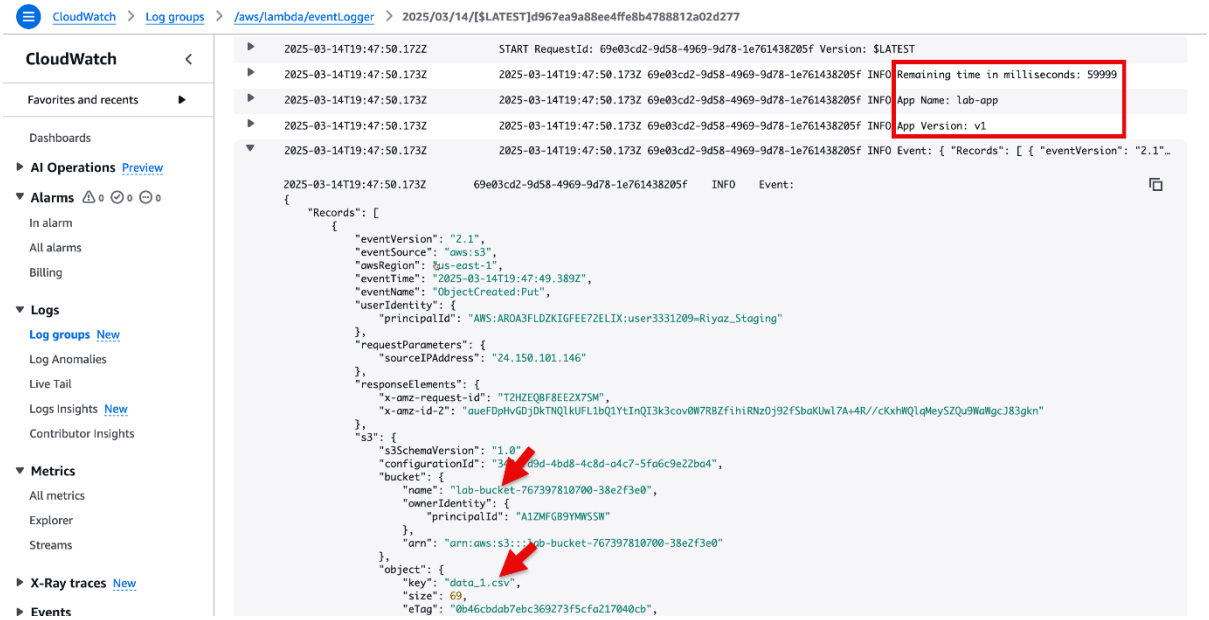


- This action will generate an **S3 PutObject event**, which will automatically trigger the Lambda function.

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.



- 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.