

AWS-Lambda- Assignment2

SHASHANK BG

Task 1 - Create an IAM Role for Lambda Execution

To allow the Lambda function to access S3 and CloudWatch, we will create an IAM role with the necessary permissions. This role ensures that Lambda has the correct access to read and write objects from S3 and log execution details to CloudWatch.

- Open the **AWS IAM Console**
- Click **Roles** in the left sidebar
- Click **Create role**
- Choose **AWS service** as the trusted entity type and select **Lambda** as the use case.
- Click **Next**

The screenshot shows the 'Create role' wizard in the AWS IAM console. The breadcrumb navigation at the top reads 'IAM > Roles > Create role'. On the left, a sidebar shows the steps: Step 1 (Select trusted entity, highlighted), Step 2 (Add permissions), and Step 3 (Name, review, and create). The main content area is titled 'Select trusted entity' and contains two sections. The 'Trusted entity type' section has five radio button options: 'AWS service' (selected, with a red arrow pointing to it), 'AWS account', 'Web identity', 'SAML 2.0 federation', and 'Custom trust policy'. The 'Use case' section has a dropdown menu for 'Service or use case' with 'Lambda' selected (indicated by a red arrow). Below the dropdown, it says 'Choose a use case for the specified service.' and lists 'Lambda' as the selected use case. At the bottom right, there are 'Cancel' and 'Next' buttons, with a red arrow pointing to the 'Next' button.

- Search for and select the following policies:
 - AWSLambdaBasicExecutionRole (for CloudWatch logs)

The screenshot shows the 'Add permissions' step of the 'Create role' wizard. The breadcrumb navigation at the top reads 'IAM > Roles > Create role'. On the left, the sidebar shows Step 1 (Select trusted entity) and Step 2 (Add permissions, highlighted). The main content area is titled 'Add permissions' and contains a section 'Permissions policies (1/1038)'. It says 'Choose one or more policies to attach to your new role.' and has a search bar with 'AWSLambdaBasicExecutionRole' entered (indicated by a red arrow). To the right of the search bar is a 'Filter by Type' dropdown set to 'All types' and a '1 match' indicator. Below the search bar is a table with columns 'Policy name', 'Type', and 'Description'. The table has one row: 'AWSLambdaBasicExecutionRole' (checked), 'AWS managed', and 'Provides write permissions to CloudW...'. At the bottom, there is a section 'Set permissions boundary - optional' and 'Cancel', 'Previous', and 'Next' buttons.

- AmazonS3FullAccess (for S3 access)

IAM > Roles > Create role

Step 1: Select trusted entity
Step 2: **Add permissions**
Step 3: Name, review, and create

Add permissions info

Permissions policies (2/1038) info

Choose one or more policies to attach to your new role.

Filter by Type: All types 1 match

<input checked="" type="checkbox"/>	Policy name <small>?</small>	Type	Description
<input checked="" type="checkbox"/>	AmazonS3FullAccess	AWS managed	Provides full access to all buckets via t...

► Set permissions boundary - optional

Cancel Previous **Next**

- Click **Next**
- Name the role LabLambdaS3ExecutionRole and click **Create role**

IAM > Roles > LabLambdaS3ExecutionRole

Identity and Access Management (IAM)

Search IAM

Dashboard

▼ Access management

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

▼ Access reports

- Access Analyzer
- External access
- Unused access
- Analyzer settings
- Credential report
- Organization activity

LabLambdaS3ExecutionRole info

Allows Lambda functions to call AWS services on your behalf.

Summary Edit

Creation date March 15, 2025, 11:02 (UTC-04:00)	ARN arn:aws:iam::211125701674:role/LabLambdaS3ExecutionRole
Last activity -	Maximum session duration 1 hour

Permissions Trust relationships Tags Last Accessed Revoke sessions

Permissions policies (2) info

You can attach up to 10 managed policies.

Filter by Type: All types

<input type="checkbox"/>	Policy name <small>?</small>	Type	Attached entities
<input type="checkbox"/>	AmazonS3FullAccess	AWS managed	1
<input type="checkbox"/>	AWSLambdaBasicExecutionRole	AWS managed	1

Task 2 - Create the Lambda Function

AWS Lambda provides an event-driven execution environment for running code without provisioning servers. In this step, we will create the Lambda function responsible for resizing images when triggered by an S3 event.

- Open the **AWS Lambda Console**
- Click **Create function**
- Select **Author from scratch**
- Enter **resizeImages** as the function name
- Choose **Node.js 22.x** as the runtime
- Expand **Change default execution role**, select **Use an existing role** and choose LabLambdaS3ExecutionRole

[Lambda](#) > [Functions](#) > [Create function](#)

☒ **Author from scratch**
Start with a simple Hello World example.

☐ **Use a blueprint**
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
☒ Node.js 22.x

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
☒ x86_64
☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

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

☐ Create a new role with basic Lambda permissions
☒ Use an existing role
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
☒ LabLambdaS3ExecutionRole

[View the LabLambdaS3ExecutionRole role](#) on the IAM console.

- Click **Create function**
- Go to the **Configuration** tab, and open **Environment variables** section.
- Add an environment variable named `DESTINATION_BUCKET` to the function. Set its value to the name of your target s3 bucket i.e. `lab-s3-destination-bucket-<random-id>`. You can get the exact name of the bucket from your S3 console.

[Lambda](#) > [Functions](#) > [resizelimages](#)

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

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

Environment variables (1) [Edit](#)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
DESTINATION_BUCKET	lab-s3-destination-bucket-211125701674-d4805160

Task 3 - Add a Lambda Layer for Image Processing

Lambda Layers allow us to package external libraries and dependencies separately from our function code. This reduces the function deployment package size, speeds up deployments, and enables code reuse across multiple Lambda functions. Instead of packaging the **Sharp** image processing library within the Lambda function, we will create a **Lambda Layer** and attach it to our function.

Understanding Lambda Layers

A Lambda Layer is an archive that contains libraries, dependencies, or even custom runtime components that can be shared across multiple Lambda functions. Instead of bundling the Sharp

image processing library inside our function's deployment package, we will create a **dedicated Lambda Layer** that can be reused, reducing function package size and improving performance.

Benefits of using a Lambda Layer for Sharp:

- **Optimized Deployment:** Reduces the size of your Lambda function package, making deployments faster.
- **Reusability:** The same layer can be used across multiple Lambda functions without reinstallation.
- **Improved Maintainability:** Allows easy updates to dependencies without modifying function code.
- **Performance Boost:** Helps Lambda cold starts by keeping the function package lightweight.

Initialize a Node.js Project and Install Sharp

- Create a directory to store the project files and navigate to it e.g. `mkdir -p ~/lab-resize-images` && `cd ~/lab-resize-images`. Then, open the project folder `lab-resize-images` in **VS Code** with code . or use any code editor of your choice.
- You can also manually create a folder named `lab-resize-images` and open it using **VS Code**.
- Run the following commands to create a Node.js project and install Sharp for multiple environments:

1. `npm init -y`

1.# Install Sharp for different CPU architectures and operating systems

2. `npm install --save sharp`

3. `npm install --cpu=x64 --os=darwin sharp`

4. `npm install --cpu=arm64 --os=darwin sharp`

5. `npm install --cpu=x64 --os=linux --libc=glibc sharp`

6. `npm install --cpu=x64 --os=linux --libc=musl sharp`

Package the Layer

Once the dependencies are installed, package them into a zip file for AWS Lambda:

1. `mkdir -p nodejs`

2. `cp -r node_modules nodejs/`

3. `zip -r layer_content.zip nodejs`

This creates a zip file `layer_content.zip` containing the necessary dependencies inside a `nodejs/` folder. AWS Lambda requires Node.js dependencies to be in this directory format inside the layer.

Deploy the Lambda Layer

Upload the packaged zip file as a Lambda Layer:

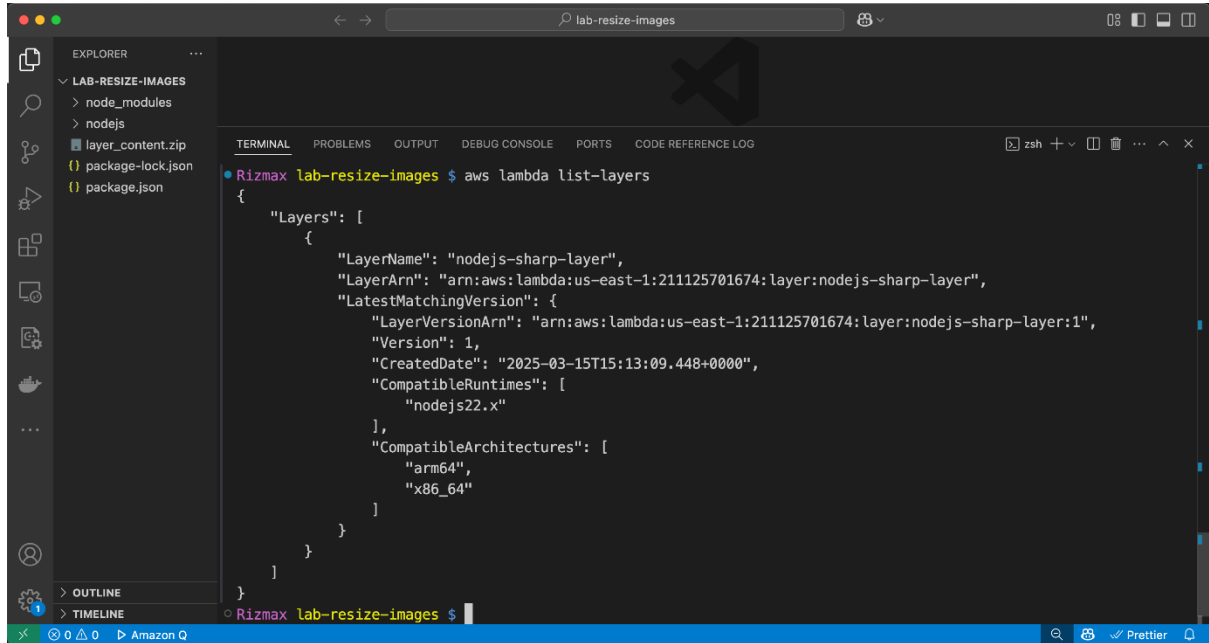
1. `aws lambda publish-layer-version --layer-name nodejs-sharp-layer \`

2. `--zip-file fileb://layer_content.zip \`

3. `--compatible-runtimes nodejs22.x \`
 4. `--compatible-architectures "arm64" "x86_64"`
- `--layer-name nodejs-sharp-layer`: Names the layer.
 - `--zip-file fileb://layer_content.zip`: Specifies the zip file containing the layer content.
 - `--compatible-runtimes nodejs22.x`: Ensures compatibility with the latest Node.js runtime.
 - `--compatible-architectures "arm64" "x86_64"`: Supports both x86 and ARM architectures.

Verify that the layer has been created successfully by listing available layers:

1.aws lambda list-layers

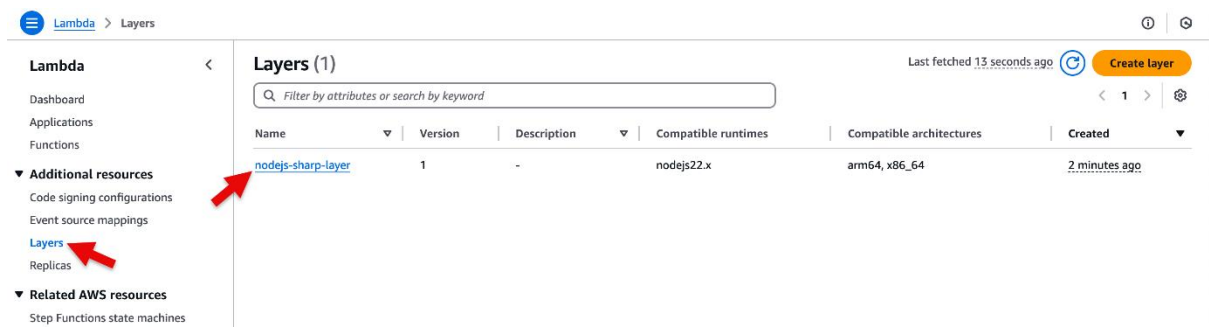


```

Rizmax lab-resize-images $ aws lambda list-layers
{
  "Layers": [
    {
      "LayerName": "nodejs-sharp-layer",
      "LayerArn": "arn:aws:lambda:us-east-1:211125701674:layer:nodejs-sharp-layer",
      "LatestMatchingVersion": {
        "LayerVersionArn": "arn:aws:lambda:us-east-1:211125701674:layer:nodejs-sharp-layer:1",
        "Version": 1,
        "CreateDate": "2025-03-15T15:13:09.448+0000",
        "CompatibleRuntimes": [
          "nodejs22.x"
        ],
        "CompatibleArchitectures": [
          "arm64",
          "x86_64"
        ]
      }
    }
  ]
}

```

You can also review the layer deployment in AWS Lambda console:



With the Lambda Layer in place, the setup is now scalable and efficient, allowing reuse across multiple functions while keeping the deployment package lightweight.

Attach the Layer to the Lambda Function

- In the **AWS Lambda Console**, open your Lambda function `resizeImages`.
- Click on the **Code** tab.
- Scroll down to the **Layers** section and click **Add a layer**.

- Select **Custom layers**.
- Choose the nodejs-sharp-layer from the dropdown list.
- Select the latest version of the layer.
- Click **Add**.

Add layer

Function runtime settings

Runtime: Node.js 22.x | Architecture: x86_64

Choose a layer

Layer source [Info](#)

Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also [create a new layer](#).

☐ AWS layers
Choose a layer from a list of layers provided by AWS.
 ☒ Custom layers
Choose a layer from a list of layers created by your AWS account.
 ☐ Specify an ARN
Specify a layer by providing the ARN.

Custom layers

Layers created by your AWS account that are compatible with your function's runtime.

nodejs-sharp-layer

Version: 1

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

Once added, the function can now use the Sharp library without having it in the deployment package.

Task 4 - Configure S3 Event Trigger

To automate the image processing workflow, we need to configure an **S3 Event Notification** that triggers the Lambda function whenever a new image is uploaded. This enables real-time image processing without requiring manual intervention.

- Scroll up to the **Function overview** section.
- Click **Add trigger**, then select **S3** as the event source.
- Choose the S3 bucket lab-s3-source-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.
- Add .jpeg as **Suffix**.
- 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

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 have 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.

Now, whenever an object is created in lab-s3-source-bucket-<random-id>, Amazon S3 will invoke the function.

Task 5 - Create and Deploy the Lambda Function Code

Instead of manually writing and editing code in the AWS Lambda console, we will develop the function locally and deploy it using the AWS CLI. This allows better version control and testing before deployment.

- Open the project folder lab-resize-images in **VS Code**
- Create a new file named index.mjs, paste in the following code and save changes: <https://github.com/rizmaxed/sls-labs-code/blob/main/lab-10-3/index.mjs>

Before deploying the function, take a moment to review the code. Note that we are referencing the sharp library as we'd normally do, however we won't be including it in our lambda deployment package as it will be referenced from the Lambda layer that we already deployed.

Also, **notice how it uses structured logging!**. Instead of simple console.log statements, we log **structured JSON objects**. This makes logs easier to **search, filter, and analyze** in CloudWatch and other monitoring tools. For example, instead of:

```
1.console.log("Fetching image from S3");
```

We do:

```
1.console.log(JSON.stringify({
2.  level: "info",
3.  requestId,
4.  message: "Fetching image from S3",
5.  bucket: bucket.name,
6.  key
```

```
7.}));
```

Now, logs are **machine-readable**, allowing **powerful filtering** like:

- **Show only errors**
- **Find logs for a specific request ID**
- **Analyze function performance over time**

This is **next-level logging**—a simple tweak with **huge benefits** in debugging and monitoring! ?

- Now, deploy this function code via AWS CLI. To do this, open the terminal using Ctrl + ` and run the following commands. Note that you must have AWS CLI installed and configured with access to your AWS account.

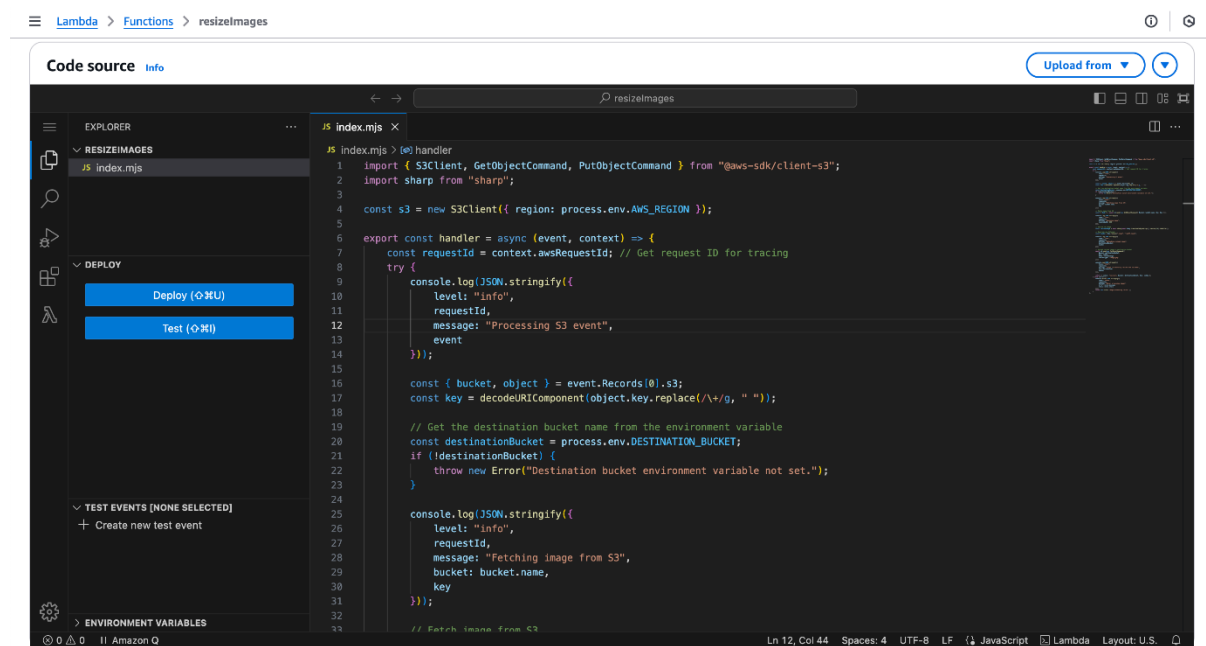
```
1.zip -r resize-images.zip index.mjs
```

```
2.BUCKET_NAME=$(aws s3 ls | awk '{print $3}' | grep '^lab-s3-source-bucket-' | head -n 1)
```

```
3.aws s3 cp resize-images.zip s3://$BUCKET_NAME/
```

```
4.aws lambda update-function-code --function-name resizImages --s3-bucket $BUCKET_NAME -  
-s3-key resize-images.zip --publish
```

This updates and deploys the function. You can also verify the upload by reviewing the function in AWS Lambda console.



Task 6 - Test the Image Processing Pipeline

To verify the correctness of our setup, we will test the Lambda function by uploading an image to S3 and checking whether it gets resized and stored in the destination bucket.

- Download a sample image:

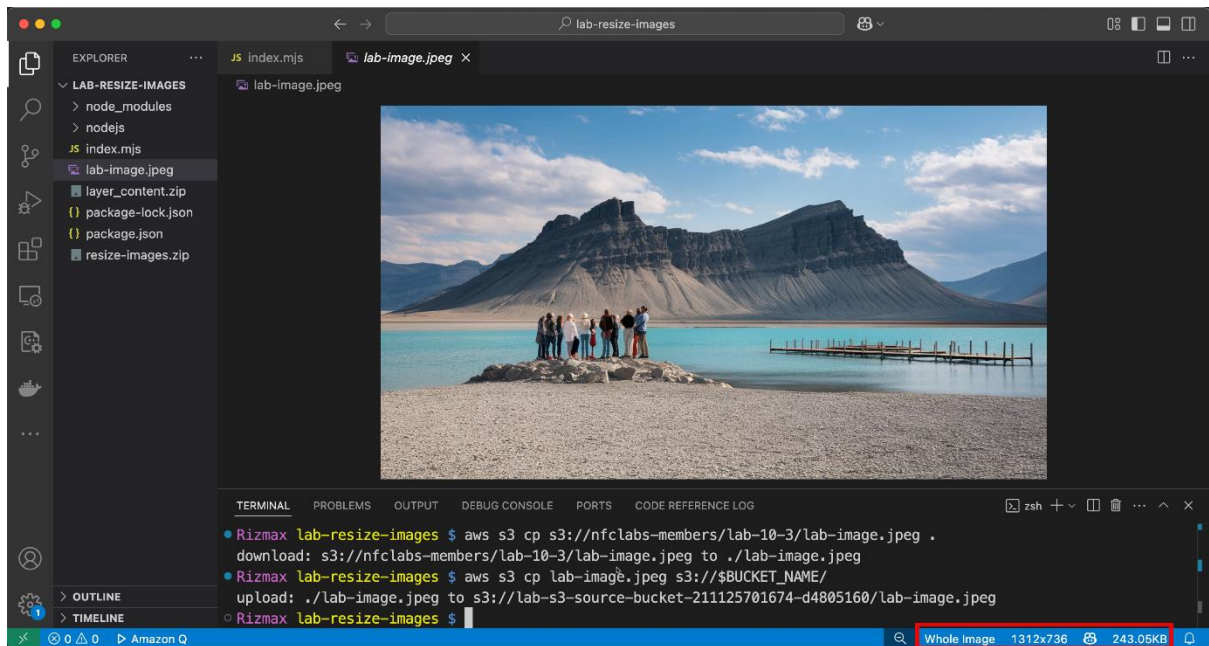
```
1. aws s3 cp s3://nflclabs-members/lab-10-3/lab-image.jpeg .
```

- Open lab-image.jpeg on your local system and note its dimensions (1312 × 736px) and file size (around 243 KB).

- Upload the image to the source S3 bucket:

1. `aws s3 cp lab-image.jpeg s3://$BUCKET_NAME/`

This triggers the Lambda function.



- Open the **AWS Lambda Console**, go to **Monitoring** → **CloudWatch Logs**, and select the latest log stream.
- Look for structured logs in JSON format:
 1. {
 2. "level": "info",
 3. "requestId": "12345678-abcd-efgh-ijkl-123456789012",
 4. "message": "Fetching image from S3",
 5. "bucket": "lab-s3-source-bucket-xyz123",
 6. "key": "lab-image.jpeg"
 7. }

CloudWatch > Log groups > /aws/lambda/resizeImages > 2025/03/15/[\$LATEST]e70cf264798948c68cac5c4b4731cecb

CloudWatch

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Filter events - press enter to search

Clear 1m 30m 1h 12h Custom UTC timezone Display

Timestamp | Message

No older events at this moment. [Retry](#)

2025-03-15T15:52:41.240Z INIT_START Runtime Version: nodejs:22.v35 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:8ce0861bacc1c3e6874...

2025-03-15T15:52:42.189Z START RequestId: d7bb2f93-c6c4-45c6-a4b5-930055c1341f Version: \$LATEST

2025-03-15T15:52:42.201Z 2025-03-15T15:52:42.201Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO {"level":"info","requestId":"d7bb2f93-c6c4-45c6-a4...

2025-03-15T15:52:42.224Z 2025-03-15T15:52:42.224Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO {"level":"info","requestId":"d7bb2f93-c6c4-45c6-a4...

2025-03-15T15:52:43.181Z 2025-03-15T15:52:43.181Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO {"level":"info","requestId":"d7bb2f93-c6c4-45c6-a4...

2025-03-15T15:52:43.961Z 2025-03-15T15:52:43.961Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO {"level":"info","requestId":"d7bb2f93-c6c4-45c6-a4...

2025-03-15T15:52:43.961Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO

{

"level": "info",

"requestId": "d7bb2f93-c6c4-45c6-a4b5-930055c1341f",

"message": "Uploading resized image",

"destinationBucket": "lab-s3-destination-bucket-21125701674-d4805160",

"newKey": "lab-image-small.jpeg"

}

2025-03-15T15:52:44.131Z 2025-03-15T15:52:44.131Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO {"level":"info","requestId":"d7bb2f93-c6c4-45c6-a4...

2025-03-15T15:52:44.131Z d7bb2f93-c6c4-45c6-a4b5-930055c1341f INFO

{

"level": "info",

"requestId": "d7bb2f93-c6c4-45c6-a4b5-930055c1341f",

"message": "Image successfully resized and uploaded",

"destinationBucket": "lab-s3-destination-bucket-21125701674-d4805160",

"newKey": "lab-image-small.jpeg"

}

2025-03-15T15:52:44.162Z END RequestId: d7bb2f93-c6c4-45c6-a4b5-930055c1341f

2025-03-15T15:52:44.163Z REPORT RequestId: d7bb2f93-c6c4-45c6-a4b5-930055c1341f Duration: 1972.77 ms Billed Duration: 1973 ms Memory Size: 128...

- From the left sidebar, open **Log Insights**.
- **CloudWatch Log Insights** allow you to filter logs efficiently. Run the following query in CloudWatch Log Insights on the Log group /aws/lambda/resizeImages:
 1. fields @timestamp, level, message, requestId
 2. | filter level="info"
 3. | sort @timestamp desc

CloudWatch > Logs Insights

CloudWatch

Log Insights

Select log groups by

Log group name Select up to 50 log groups Browse log groups

/aws/lambda/resizeImages X Clear all

1 fields @timestamp, level, message, requestId

2 | filter level="info"

3 | sort @timestamp desc

Query generator

Run query Cancel Save History

Logs Insights QL query can run for maximum of 60 minutes.

Completed. Query executed for 1 log groups.

Logs (5) Patterns (-) Visualization

Investigate Export results Add to dashboard

Showing 5 of 5 records matched 12 records (3.3 kB) scanned in 1.0s @ 12 records/s (3.3 kB/s)

Hide histogram

@timestamp level message requestId

1 2025-03-15T15:52:44.1... info Image successfully resized and uploaded d7bb2f93-c6c4-45c6-a4b5-930055c1341f

2 2025-03-15T15:52:43.9... info Uploading resized image d7bb2f93-c6c4-45c6-a4b5-930055c1341f

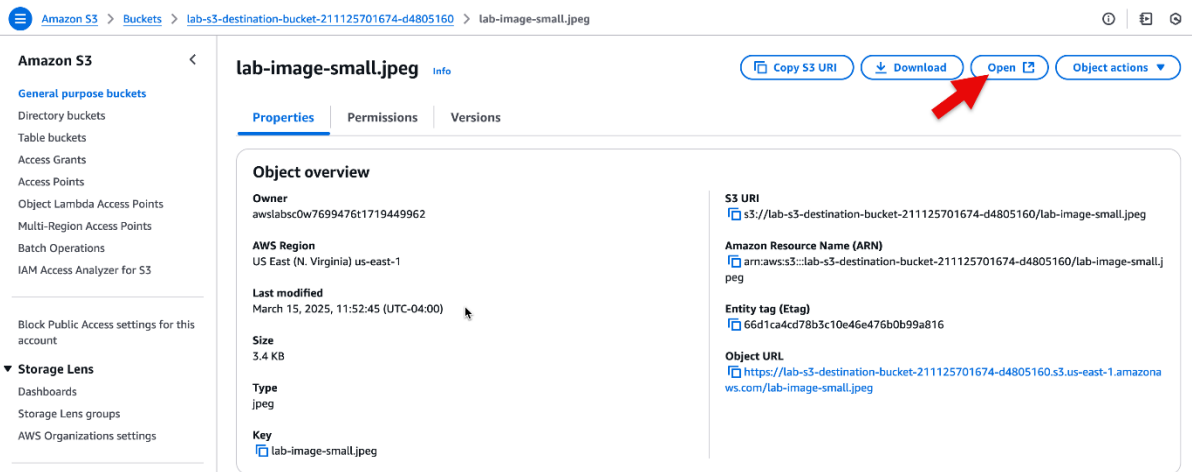
3 2025-03-15T15:52:43.1... info Resizing image d7bb2f93-c6c4-45c6-a4b5-930055c1341f

4 2025-03-15T15:52:42.2... info Fetching image from S3 d7bb2f93-c6c4-45c6-a4b5-930055c1341f

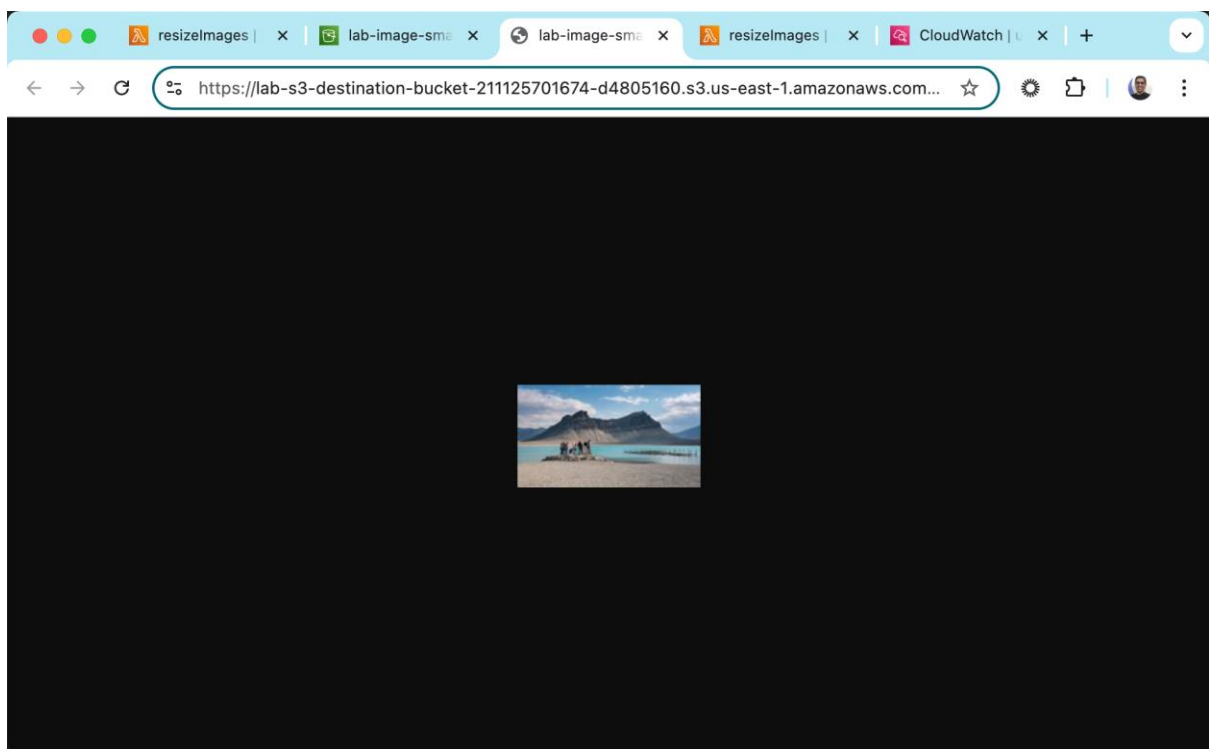
5 2025-03-15T15:52:42.2... info Processing S3 event d7bb2f93-c6c4-45c6-a4b5-930055c1341f

- To display only errors, you could use:
 1. fields @timestamp, level, message, requestId
 2. | filter level="error"
 3. | sort @timestamp desc

- This makes debugging, troubleshooting, and observability a whole lot easier ?
- Open the lab-s3-destination-bucket-<random-id> bucket in the **S3 Console** and look for the resized image with -small.jpeg in its name.



- Open the new image from the destination bucket and check its new dimensions and file size. It should now be 150px wide and significantly smaller.



Task 7 - Clean Up

To avoid incurring charges, we will clean up all created resources.

- Open the **AWS Lambda Console** and delete the resizImages function if it is no longer needed.
- Open the **IAM Console**, go to **Roles**, and delete LabLambdaS3ExecutionRole.
- Open the **S3 Console**. Empty and delete the lab-s3-source-bucket-<random-id> and lab-s3-destination-bucket-<random-id> buckets.

- Open **AWS Lambda Layers** and delete nodejs-sharp-layer.
- Go to the **CloudFormation** console and delete the lab-stack-10-3 stack to remove all lab infrastructure.