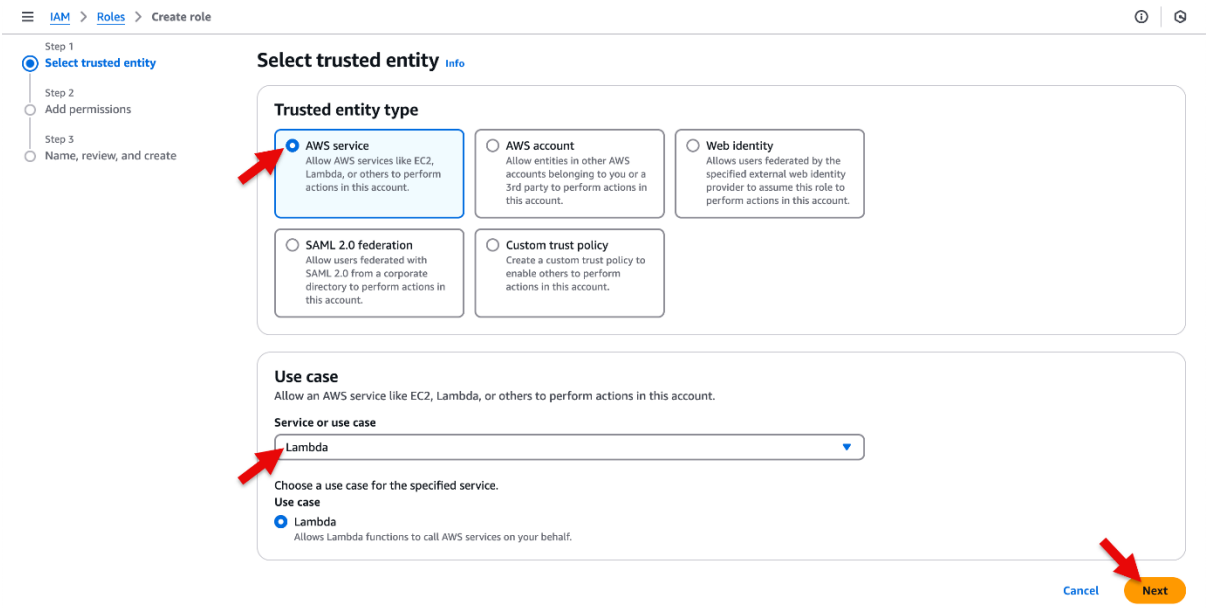
AWS-Lambda- Assignment2
SHASHANK BG

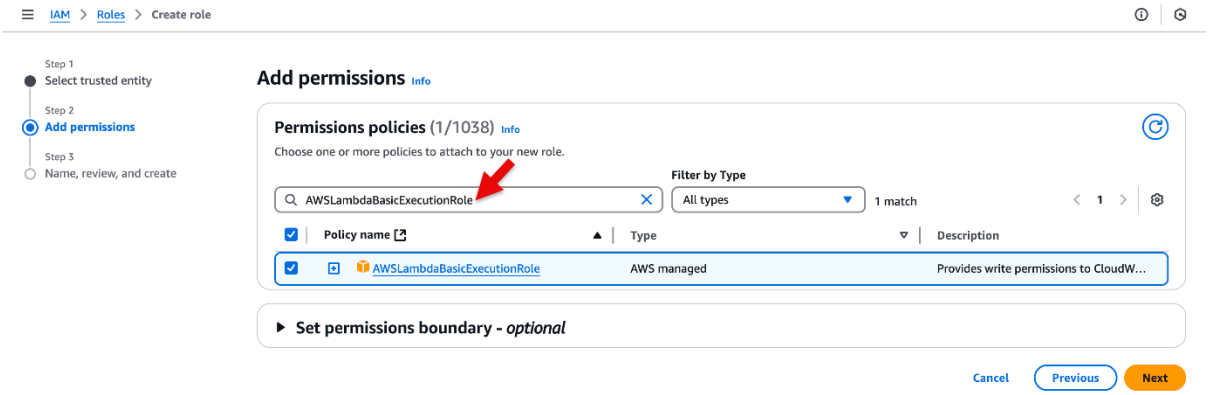
Task 1 - Create an IAM Role for Lambda Execution

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.

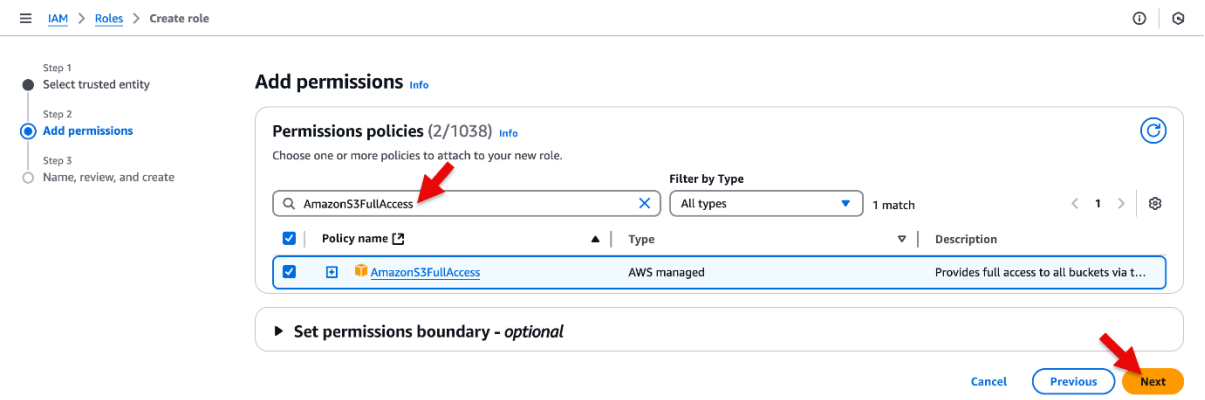
- Open the **AWS IAM Console**
- Click **Roles** in the left sidebar
- Click **Create role**
- Choose **AWS service** as the trusted entity type and select **Lambda** as the use case.
- Click **Next**



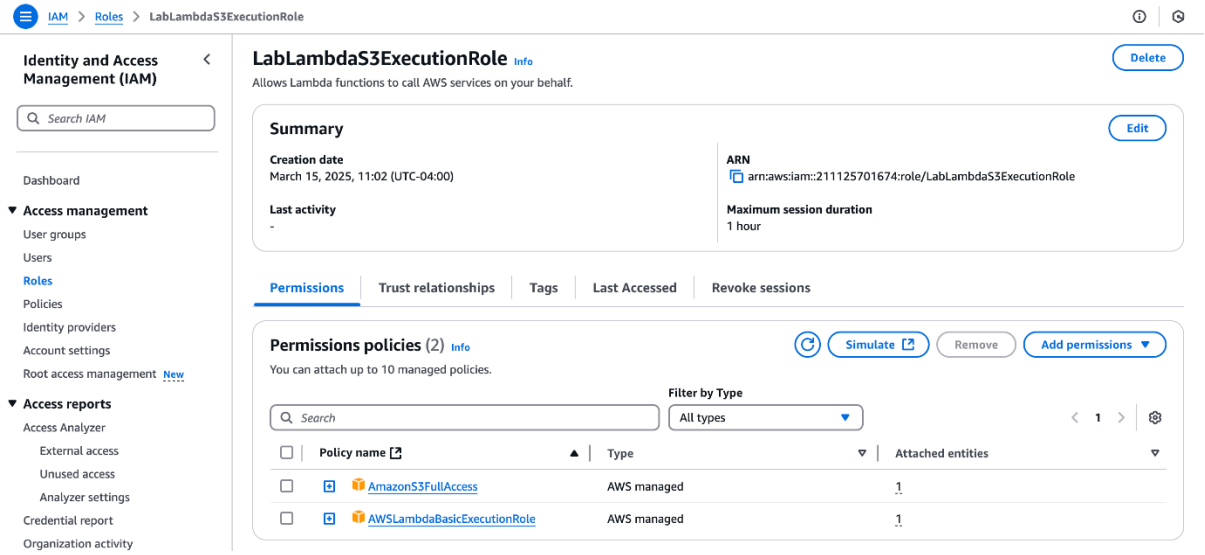
- Search for and select the following policies:
    - AWSLambdaBasicExecutionRole (for CloudWatch logs)



- AmazonS3FullAccess (for S3 access)



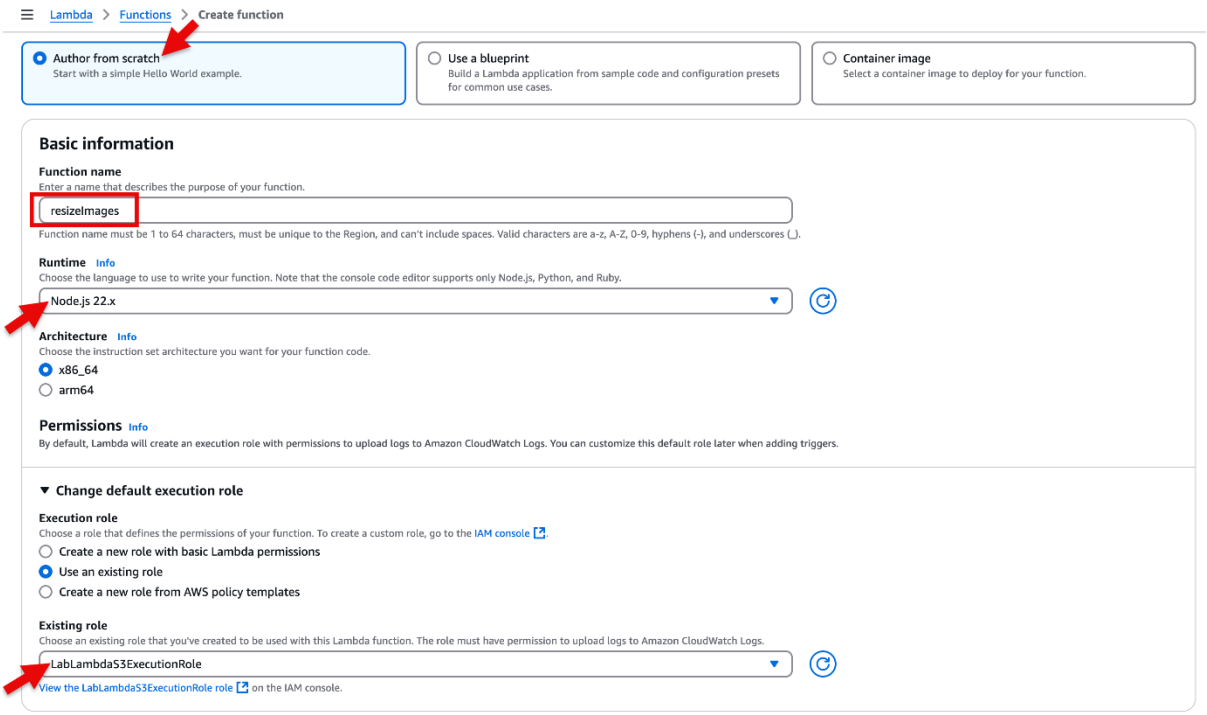
- Click **Next**
- Name the role LabLambdaS3ExecutionRole and click **Create role**



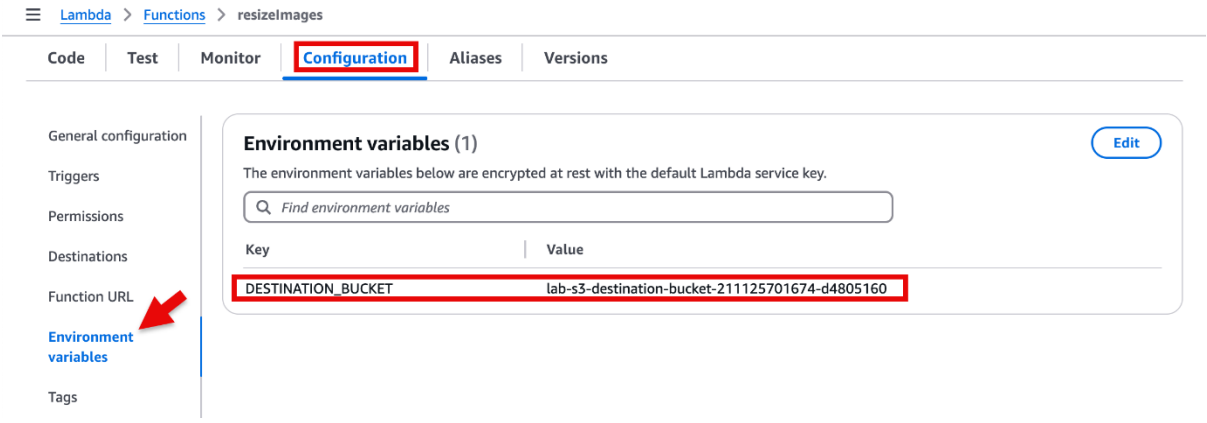
Task 2 - Create the Lambda Function

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



- 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.



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:

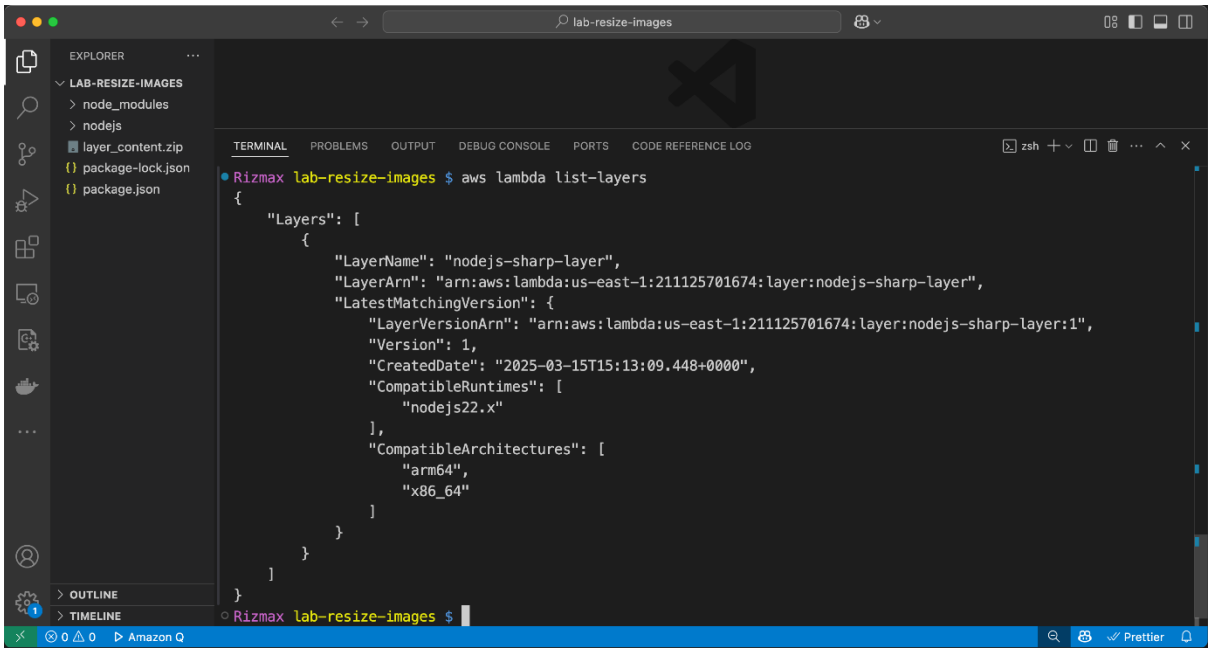
1. aws lambda publish-layer-version --layer-name nodejs-sharp-layer \
2. --zip-file fileb://layer_content.zip \

3. --compatible-runtimes nodejs22.x \
4. --compatible-architectures "arm64" "x86_64"

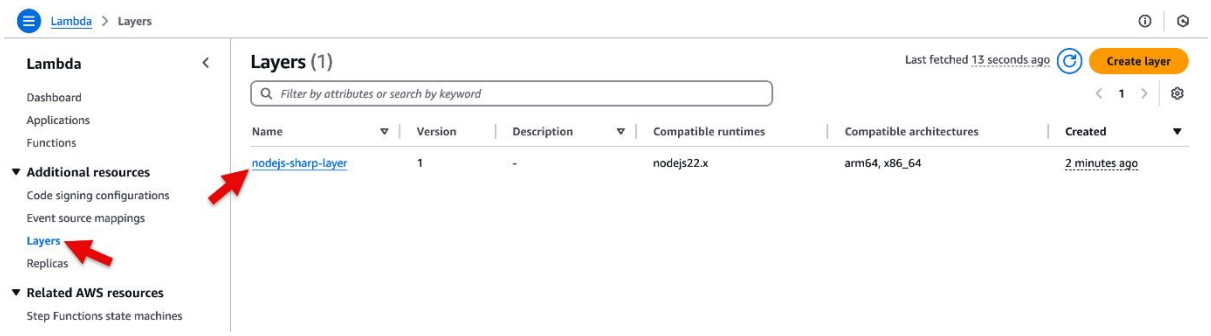
- --layer-name nodejs-sharp-layer: Names the layer.
- --zip-file fileb://layer_content.zip: Specifies the zip file containing the layer content.
- --compatible-runtimes nodejs22.x: Ensures compatibility with the latest Node.js runtime.
- --compatible-architectures "arm64" "x86_64": Supports both x86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

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

1. aws lambda list-layers



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

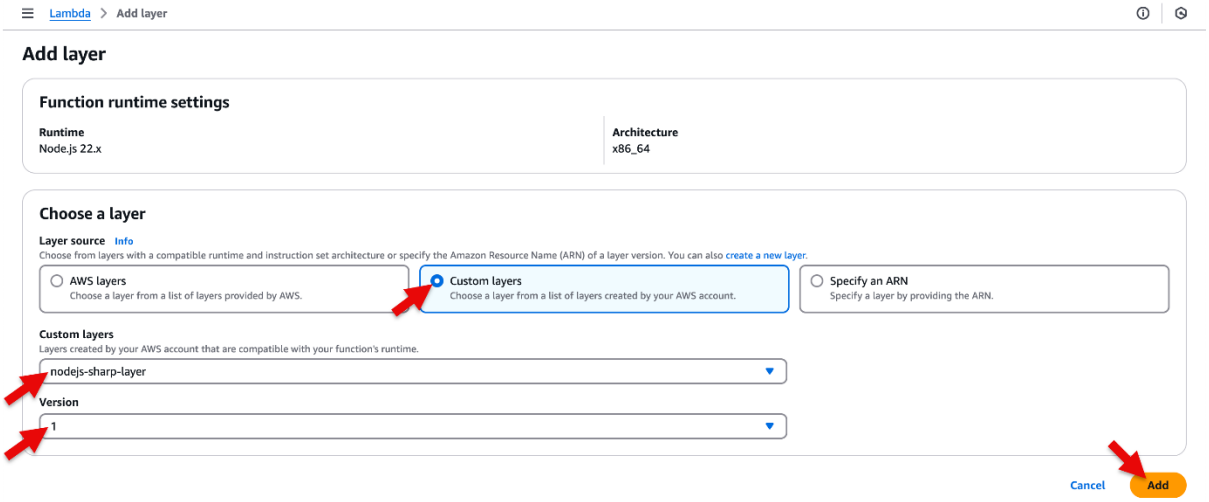


With the Lambda Layer in place, the setup is now scalable and efficient, allowing reuse across multiple functions while keeping the deployment package lightweight.

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**.

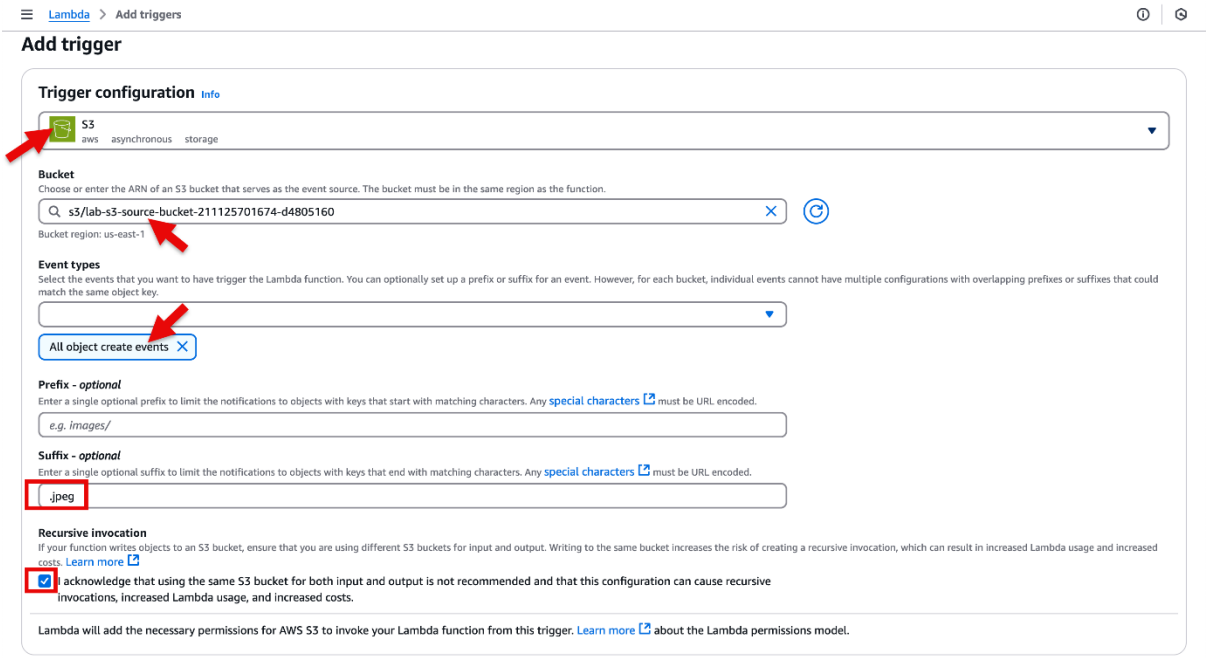


Once added, the function can now use the Sharp library without having it in the deployment package.

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.



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:

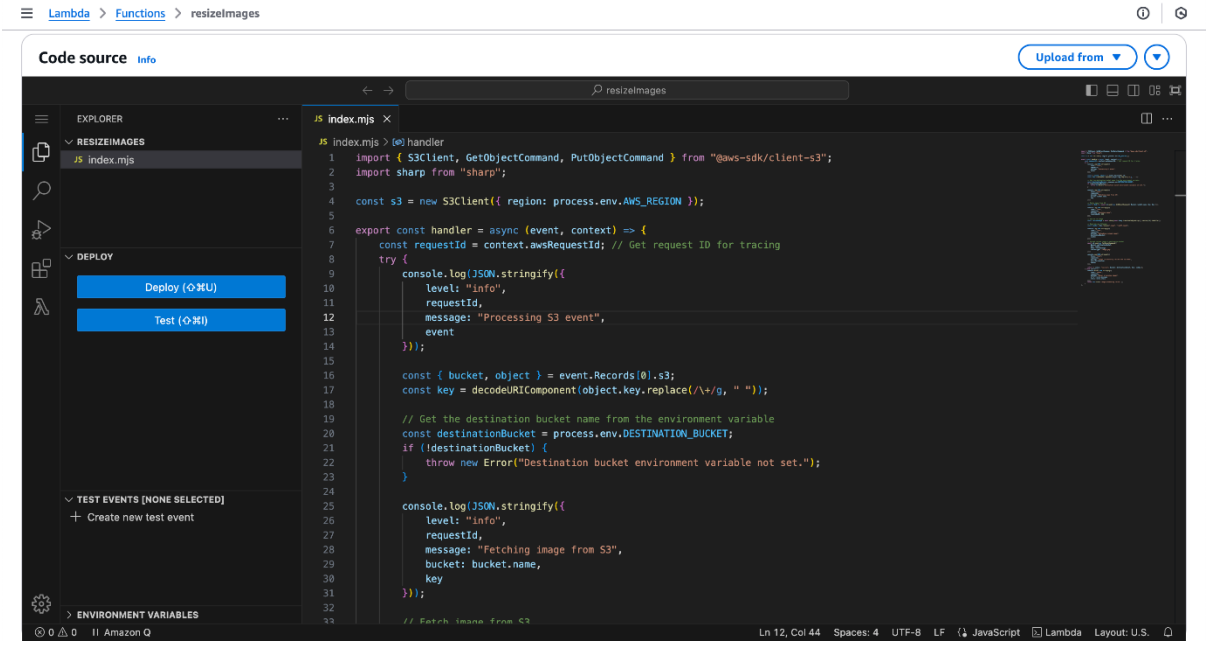
- **Show only errors**
- **Find logs for a specific request ID**
- **Analyze function performance over time**

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 resizeImages --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.

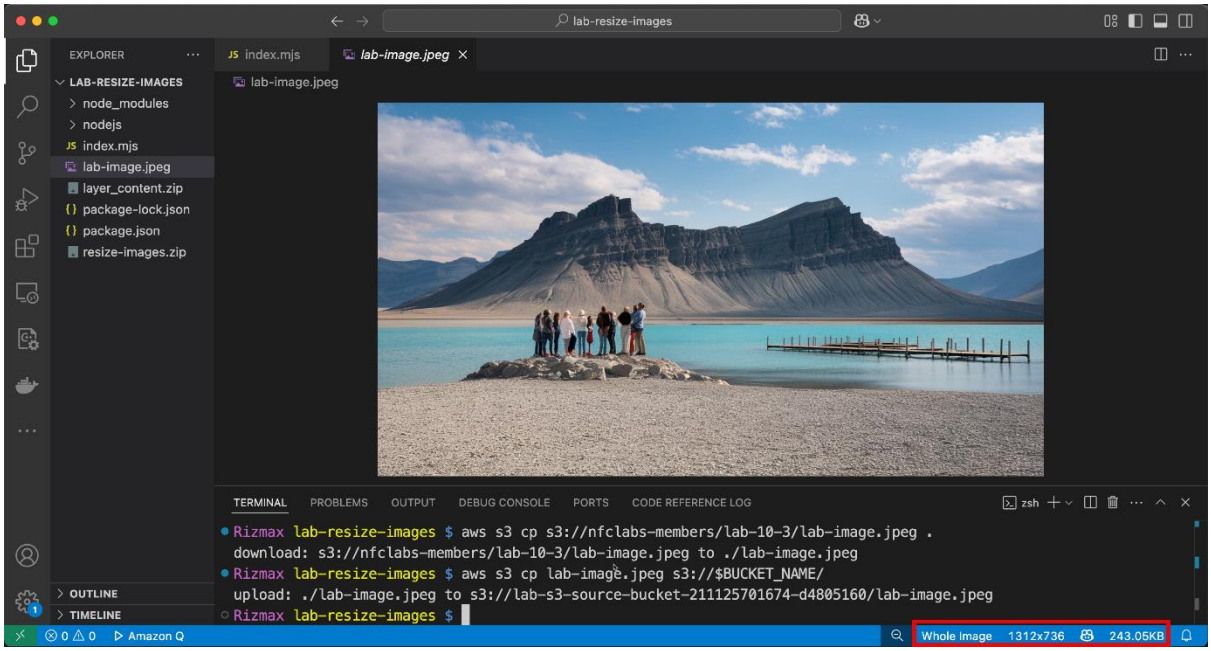
- Download a sample image:

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

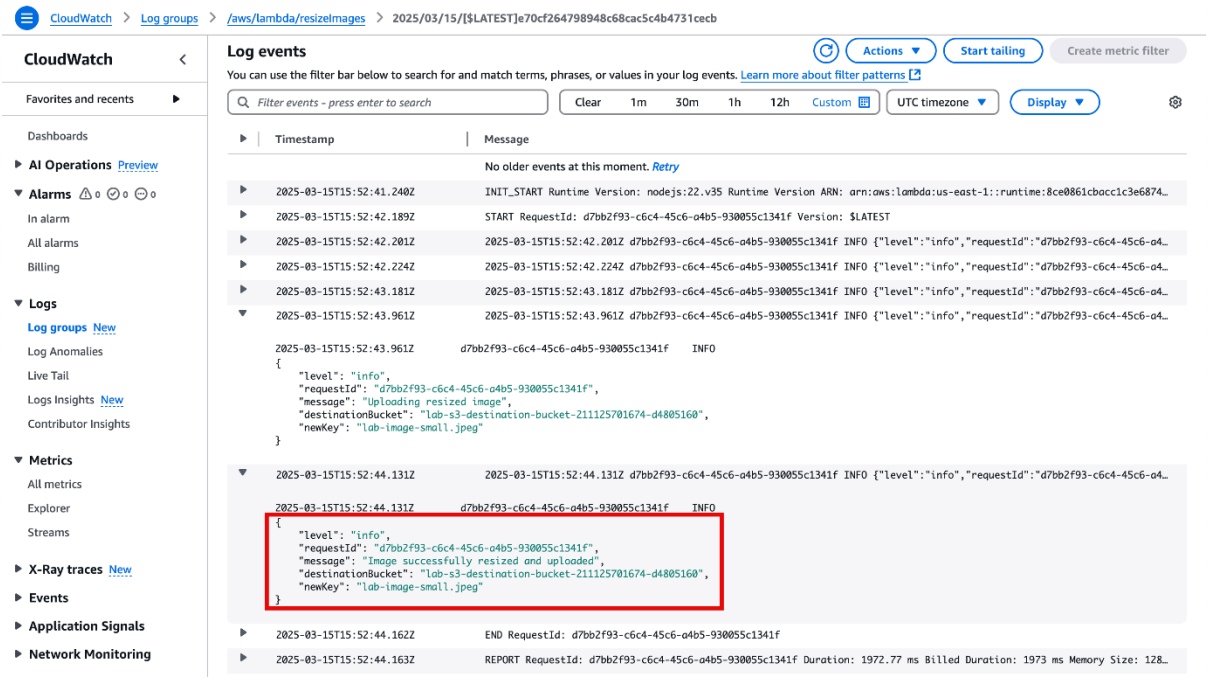
- Open lab-image.jpeg on your local system and note its dimensions (1312 × 736px) and file size (around 243 KB).

- 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. }

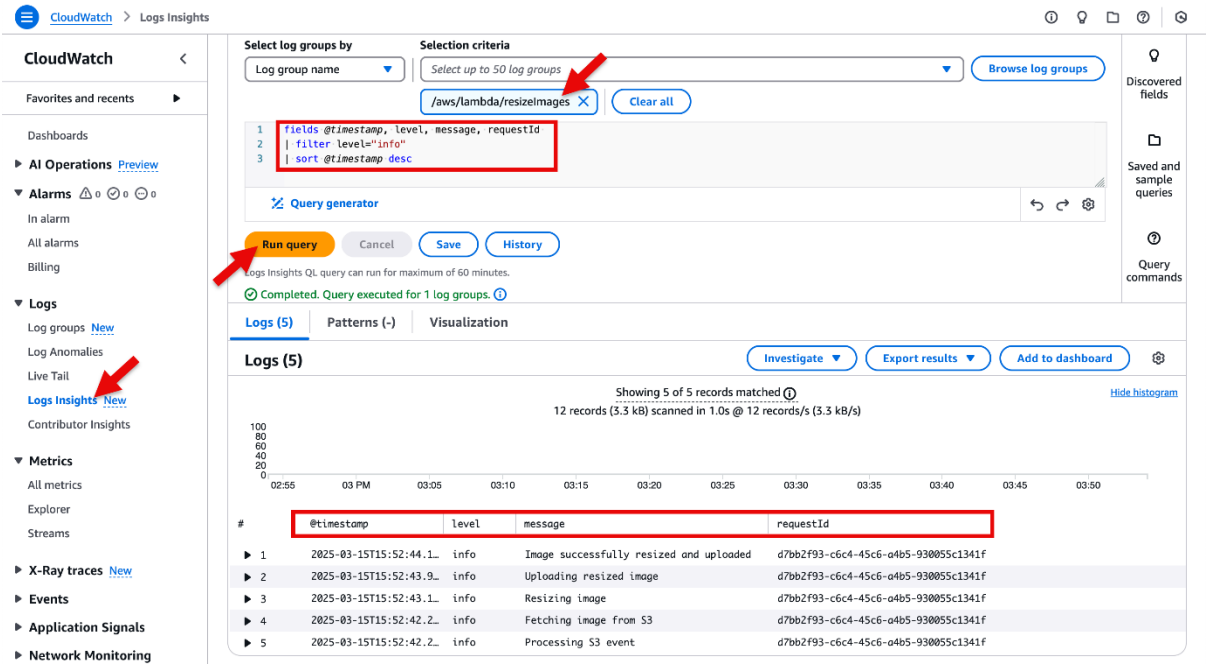


- From the left sidebar, open **Log Insights**.
- **CloudWatch Log Insights** allow you to filter logs efficiently. Run the following query in CloudWatch Log Insights on the Log group /aws/lambda/resizeImages:

1.fields @timestamp, level, message, requestId

2.| filter level="info"

3.| sort @timestamp desc



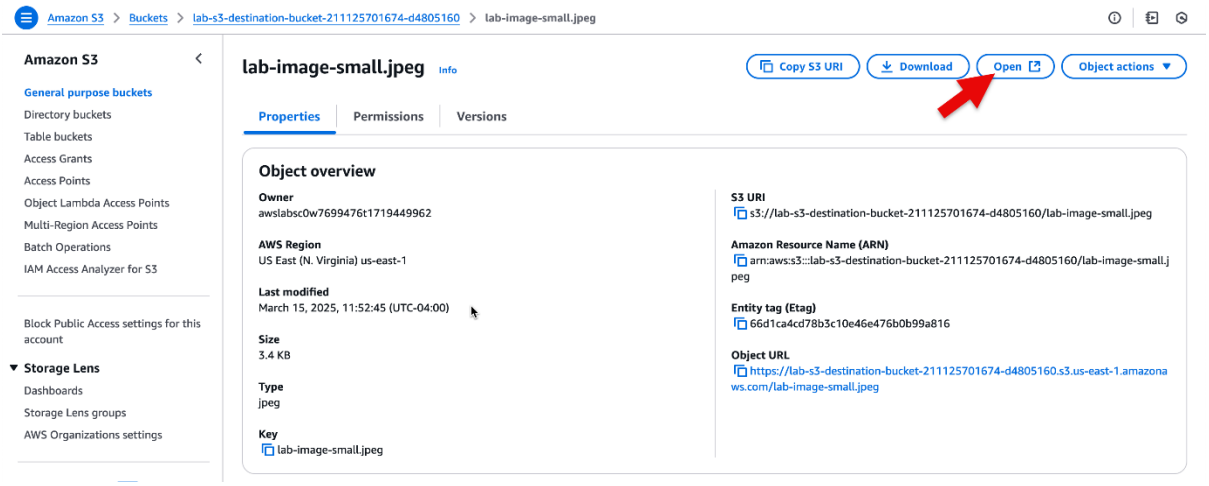
- To display only errors, you could use:

1.fields @timestamp, level, message, requestId

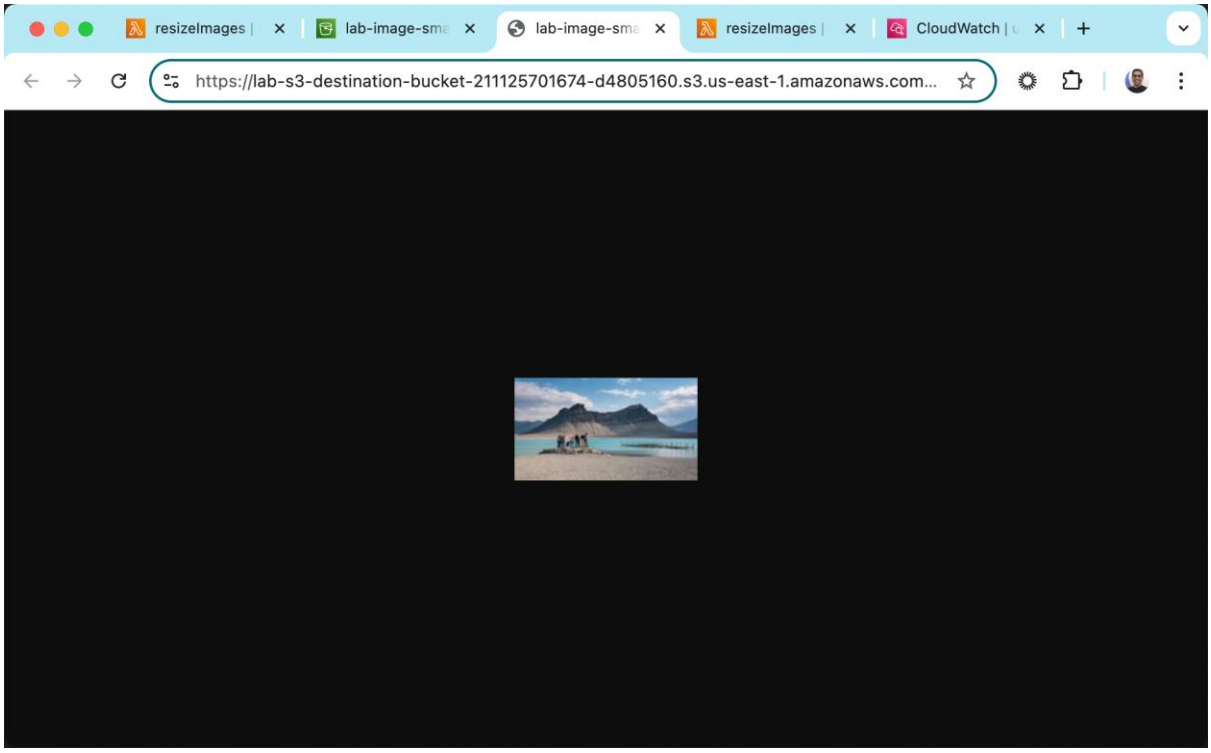
2.| filter level="error"

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 resizeImages 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.