

## EXPERIMENT 4

**Aim:** To study and Implement Platform as a Service using AWS/Microsoft Azure.

**Objective:**

- Describe microservices and Serverless concepts
- Discuss event-driven architectures
- Describe the features and benefits of AWS Lambda functions
- Discuss how to configure AWS Lambda functions
- Identify how to monitor AWS Lambda functions
- Describe best practices for working with AWS Lambda
- Identify the features and benefits of additional AWS Serverless services

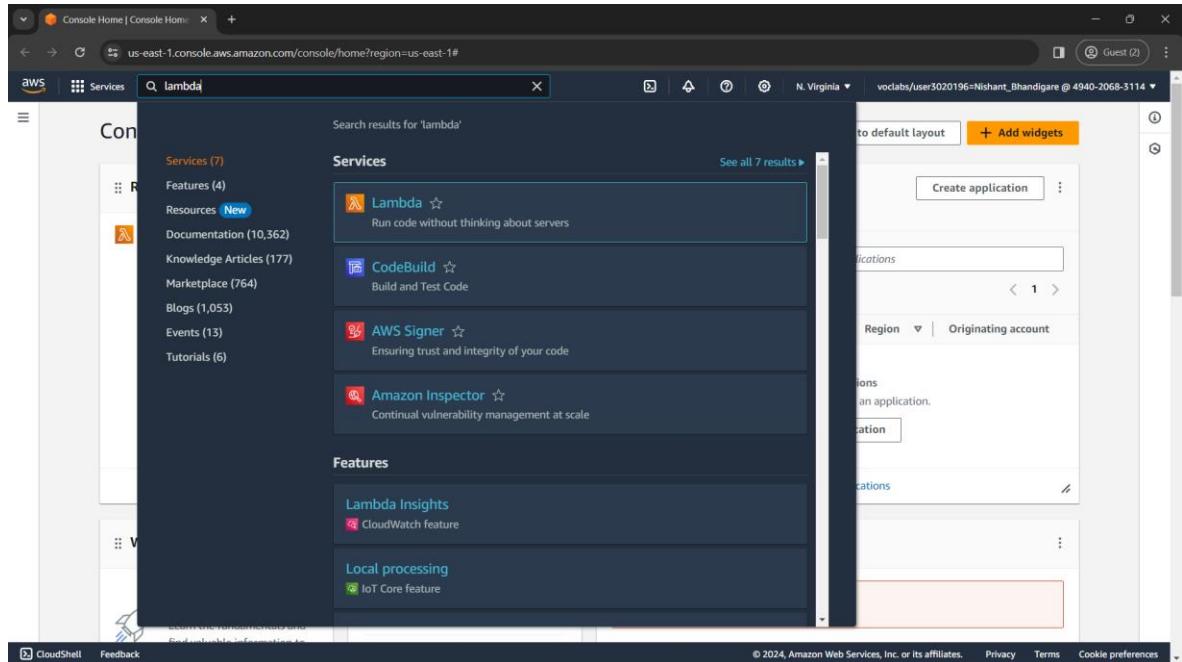
**Theory:** To embark on a study and implementation of Platform as a Service (PaaS) using AWS or Microsoft Azure, it's essential to grasp the foundational concepts and functionalities inherent in these cloud platforms. PaaS provides developers with a complete platform for building, deploying, and managing applications without the complexity of managing underlying infrastructure. Both AWS and Microsoft Azure offer robust PaaS solutions tailored to various application development needs. The theory behind this endeavor revolves around understanding the core components of PaaS offerings, such as development tools, middleware, database management systems, and runtime environments, provided by AWS and Azure. These platforms offer services like AWS Elastic Beanstalk, AWS Lambda, Azure App Service, and Azure Functions, enabling developers to focus on application development without worrying about server management, scalability, or infrastructure maintenance. Key concepts include selecting the appropriate PaaS services based on application requirements, configuring and deploying applications, integrating with other cloud services, and monitoring application performance. Through this study and implementation, participants gain insights into the agility, scalability, and efficiency offered by PaaS solutions, empowering them to accelerate application development and deployment while reducing operational overhead. This experiment serves as a practical exploration of cloud-native development paradigms and equips participants with the skills to leverage AWS or Microsoft Azure effectively for building and managing modern applications in the cloud.

**Implementation And Output:**

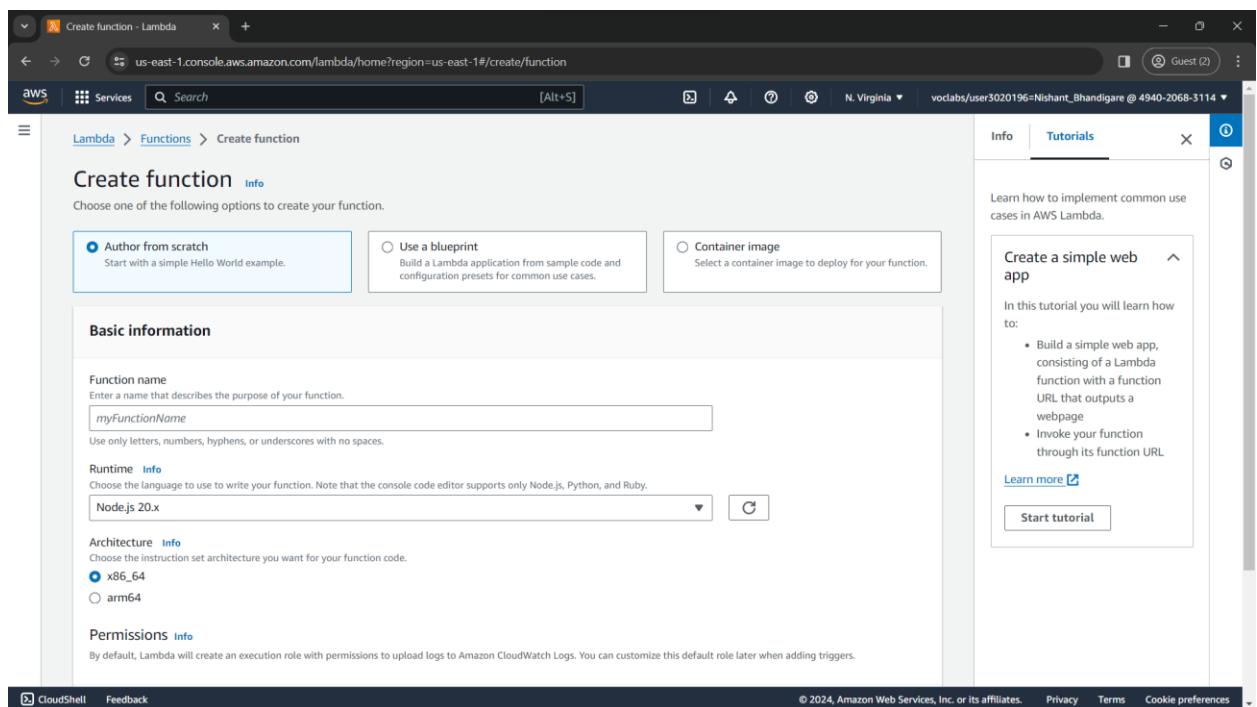
**Task 1: Creating a Lambda function**

---

1. In the **AWS Management Console**, on the **Services** menu, enter **Lambda**. From the search results, choose **Lambda**.



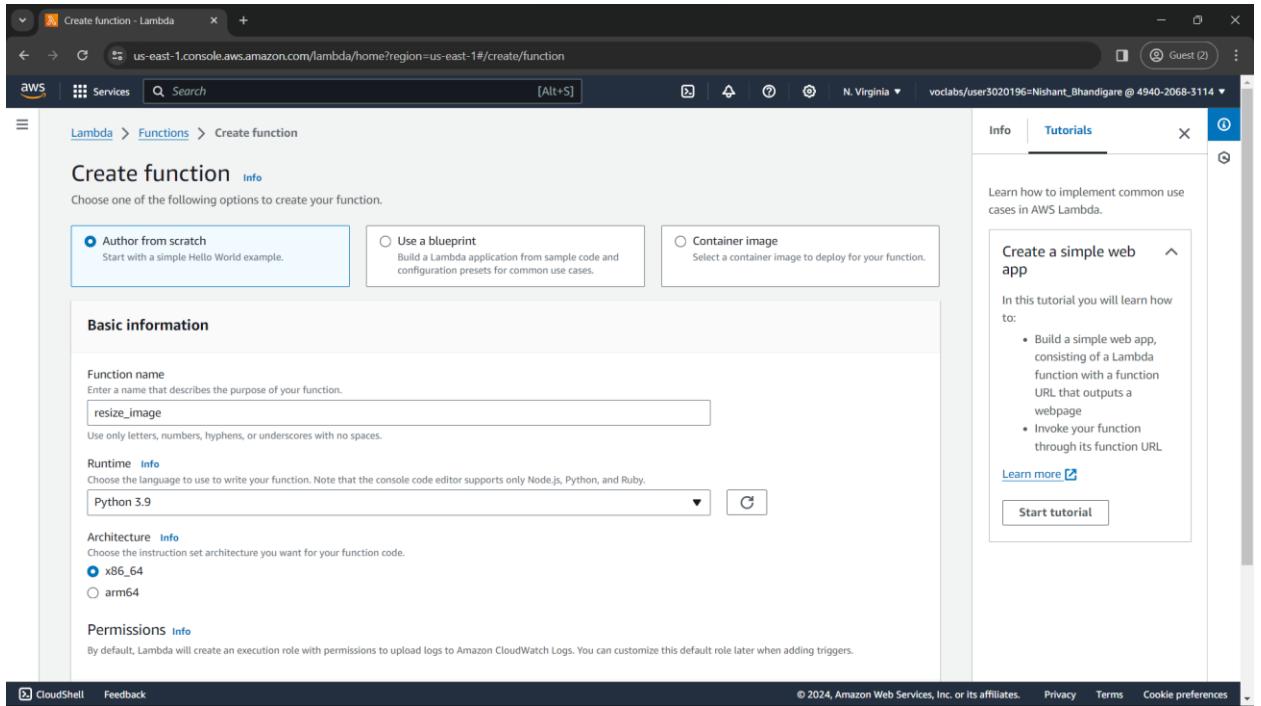
## 2. Choose Create Function.



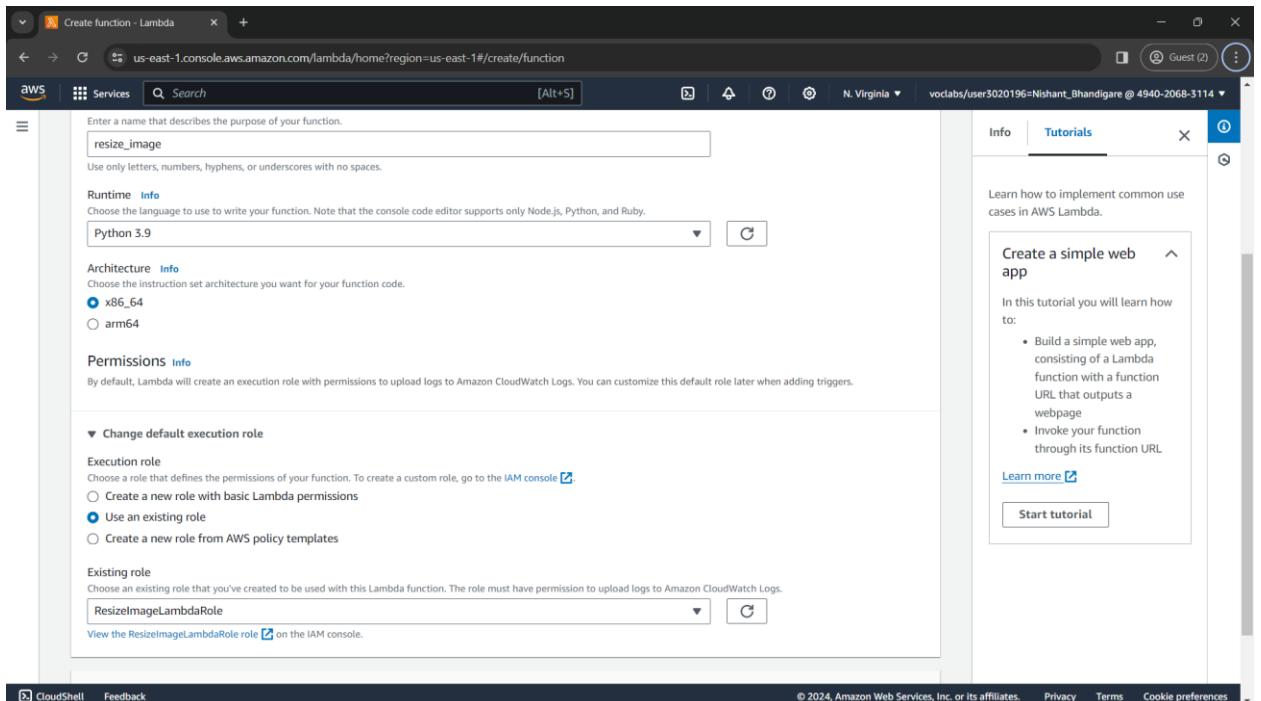
Edit the following:

**Function name:** `resize_image`

## 3. Runtime: Python3.9



4. Expand the **Change default execution role** section. Choose **Use an existing role**. In the dropdown list for **Existing Role**, choose **ResizeImageLambdaRole**.



5. Choose **Create function**.
6. Scroll to the **Layers** section.

Successfully created the function **resize\_image**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

**Code properties**

Package size: 299.0 byte    SHA256 hash: HAPq9EReJVECSgLavtc/gyd5vZtd9eiUGF932t0jBxY=    Last modified: February 29, 2024 at 02:34 PM GMT+5:30

**Runtime settings**

Runtime: Python 3.9    Handler: lambda\_function.lambda\_handler    Architecture: x86\_64

**Layers**

There is no data to display.

**Create a simple web app**

In this tutorial you will learn how to:

- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

[Learn more](#) [Start tutorial](#)

7. Choose **Add a layer**.
8. Choose **Custom layers**.
9. From the **Custom layers** dropdown list, choose **PillowPythonLambdaLayer**.
10. From the **Version** dropdown list, choose **1** or the option with the highest version number.
11. Choose **Add**.

**Add layer**

**Function runtime settings**

Runtime: Python 3.9    Architecture: x86\_64

**Choose a layer**

**Layer source**

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](#).

**Custom layers** Choose a layer from a list of layers created by your AWS account or organization.

**AWS layers** Choose a layer from a list of layers provided by AWS.

**Specify an ARN** Specify a layer by providing the ARN.

**Custom layers**

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

PillowPythonLambdaLayer

**Version**

2

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

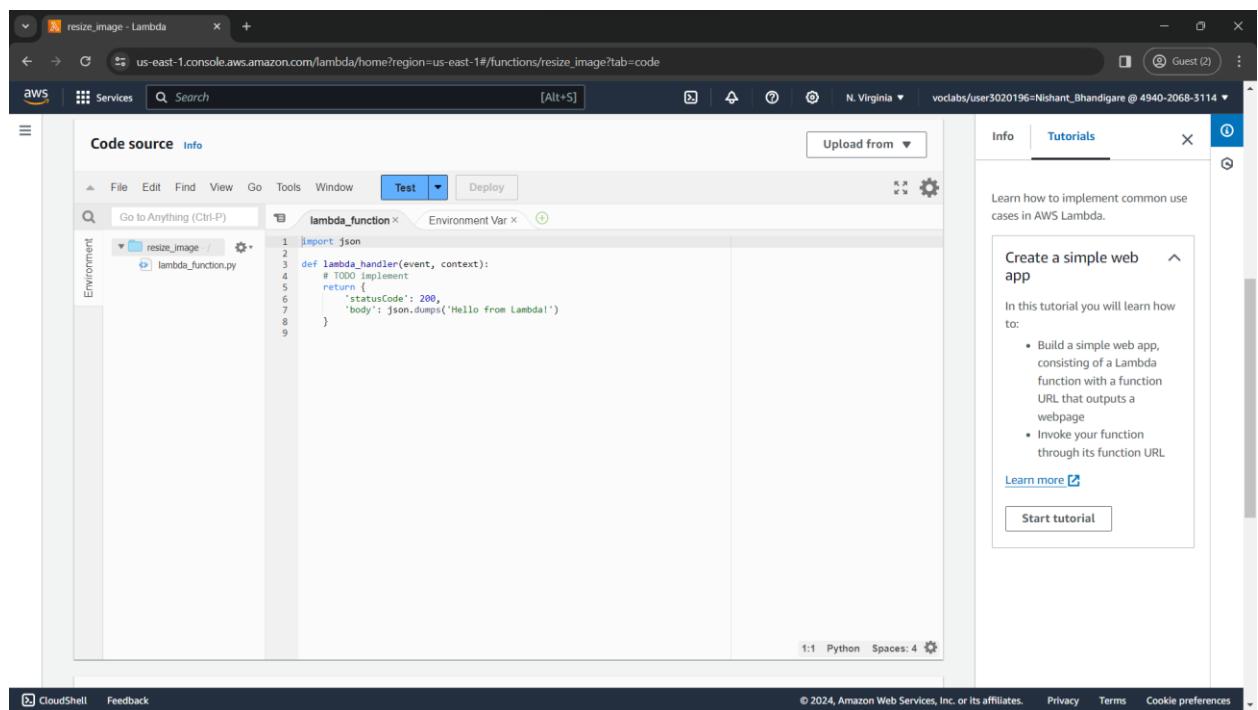
**Create a simple web app**

In this tutorial you will learn how to:

- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

[Learn more](#) [Start tutorial](#)

12. Scroll to the **Code source** section of the page.



13. Copy the code below, and replace the default source code in the `lambda_function.py` file with it.

The screenshot shows the AWS Lambda console interface. In the center, there's a code editor window titled 'lambda\_function.py'. The code is a Python script for resizing images using the AWS Lambda service. The code imports boto3, os, sys, uuid, and PIL's Image module. It defines a function 'resize\_image' that takes 'image\_path' and 'resized\_path' as parameters. Inside this function, it opens the image, resizes it by half, and saves it to 'resized\_path'. It also defines a 'lambda\_handler' function that processes S3 events, downloads the image from S3, resizes it, and uploads it back to a different S3 bucket. A sidebar on the right provides a tutorial on creating a simple web app.

```
import boto3
import os
import sys
import uuid
from urllib.parse import unquote_plus
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)

def lambda_handler(event, context):
    print('Begin resizing image')
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])
        print(key)
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}'.format(uuid.uuid4(), tmpkey)
        upload_path = '/tmp/resized-{}'.format(tmpkey)
        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, os.environ['RESIZED_BUCKET'], key)
    print('Resizing image complete')
```

```
import boto3

import os

import sys

import uuid

from urllib.parse import unquote_plus

from PIL import Image

import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):

    with Image.open(image_path) as image:

        image.thumbnail(tuple(x / 2 for x in image.size))

        image.save(resized_path)

def lambda_handler(event, context):
```

```
print('Begin resizing image')

for record in event['Records']:

    bucket = record['s3']['bucket']['name']

    key = unquote_plus(record['s3']['object']['key'])

    print(bucket)

    print(key)

    tmpkey = key.replace('/', '')

    download_path = '/tmp/{ }{ }'.format(uuid.uuid4(), tmpkey)

    upload_path = '/tmp/resized-{ }'.format(tmpkey)

    s3_client.download_file(bucket, key, download_path)

    resize_image(download_path, upload_path)

    s3_client.upload_file(upload_path, os.environ['RESIZED_BUCKET'], key)

    print('Resizing image complete')
```

14. To deploy your Lambda function, choose **Deploy**.

15. Choose the **Configuration** tab, and then choose **General configuration**. Choose **Edit**.

The screenshot shows the AWS Lambda console with the URL [us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/resize\\_image?tab=configure](https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/resize_image?tab=configure). The left sidebar has 'General configuration' selected. The main panel displays the 'General configuration' section with the following details:

Description	Memory	Ephemeral storage
-	128 MB	512 MB

Below this, the 'Timeout' is set to '0 min 3 sec' and 'SnapStart' is set to 'None'. On the right side, there is a 'Tutorials' sidebar with a 'Create a simple web app' section containing a list of steps and a 'Start tutorial' button.

16. In the **Memory** field, enter **512 MB**. Then, choose **Save**.

The screenshot shows the same AWS Lambda configuration page after the changes were saved. The 'Memory' value in the 'General configuration' table has been updated to '512 MB'. The rest of the configuration and the sidebar remain the same.

The screenshot shows the AWS Lambda console interface. The URL is [us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/resize\\_image/edit/basic-settings?tab=configure](https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/resize_image/edit/basic-settings?tab=configure). The left navigation bar shows 'Lambda > Functions > resize\_image > Edit basic settings'. The main content area is titled 'Edit basic settings' and contains the 'Basic settings' tab. Configuration options include:

- Memory**: Set to 512 MB.
- Ephemeral storage**: Set to 512 MB.
- SnapStart**: Set to None.
- Timeout**: Set to 3 seconds.

The right sidebar features a 'Tutorials' section titled 'Create a simple web app' with a list of steps:

- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

A 'Start tutorial' button is available.

This screenshot is identical to the one above, showing the 'Edit basic settings' page for the 'resize\_image' function. The configuration and sidebar are the same.

17. At the top of these instructions, choose Details. Next to AWS, choose Show. Copy the value next to **ResizedBucketName**. Use it as the value in step 21.

The screenshot shows a browser window with multiple tabs open, including 'Getting Started with Serverless' and 'Workbench - Vocareum'. The main content area displays a Lambda function code editor. The code is as follows:

```
EN-US
download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
upload_path = '/tmp/resized-{}'.format(tmpkey)
s3_client.download_file(bucket, key, download_path)
resize_image(download_path, upload_path)
s3_client.upload_file(upload_path, os.environ['RESIZED_BUCKET'], key)
print('Resizing image complete')
```

Below the code, a series of numbered steps provide instructions for deploying the function:

18. To deploy your Lambda function, choose **Deploy**.
19. Choose the **Configuration** tab, and then choose **General configuration**. Choose **Edit**.
20. In the **Memory** field, enter **512 MB**. Then, choose **Save**.
21. At the top of these instructions, choose **Details**. Next to AWS, choose **Show**. Copy the value next to **ResizedBucketName**. Use it as the value in step 25.
22. In the **AWS Management Console**, choose **Environment variables**.
23. Choose **Edit**.
24. Choose **Add environment variable**.
25. Enter the following values:  
Key: **RESIZED\_BUCKET**  
Value: Enter the value that you retrieved in step 21.
26. Choose **Save**.

You have created a Lambda function.

#### Task 2: Configuring an Amazon S3 trigger to invoke a Lambda function

The screenshot shows a browser window with multiple tabs open, including 'Getting Started with Serverless' and 'Workbench - Vocareum'. The main content area displays the 'Credentials' panel. It includes sections for 'Cloud Access' (AWS CLI, Cloud Labs) and 'Cloud Labs' session details. Below these are buttons for 'SSH key' (Show, Download PEM, Download PPK) and 'AWS SSO' (Download URL). A table lists two environment variables:

ResizedBucketName	c82801a176319615840838t1w49402068311-resizedbucket-nmtz2b2aaqxg
OriginalBucketName	c82801a176319615840838t1w4940206831-originalbucket-a601dyvrfifri

You have created a Lambda function.

#### Task 2: Configuring an Amazon S3 trigger to invoke a Lambda function

18. In the **AWS Management Console**, choose **Environment variables**.

The screenshot shows the AWS Lambda console for a function named 'resize\_image'. The 'Configuration' tab is selected. On the left, a sidebar lists various configuration options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables (which is currently selected), Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, and RDS databases. The main panel displays the 'Environment variables' section, which includes a search bar and a table with one row: 'No environment variables'. A note below states 'No environment variables associated with this function.' At the bottom right of the table is an 'Edit' button. To the right of the main panel, there is a sidebar titled 'Tutorials' with a section for 'Create a simple web app'. It includes a list of steps: 'Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage', 'Invoke your function through its function URL', and a 'Start tutorial' button.

19. Choose **Edit**.

20. Choose **Add environment variable**.

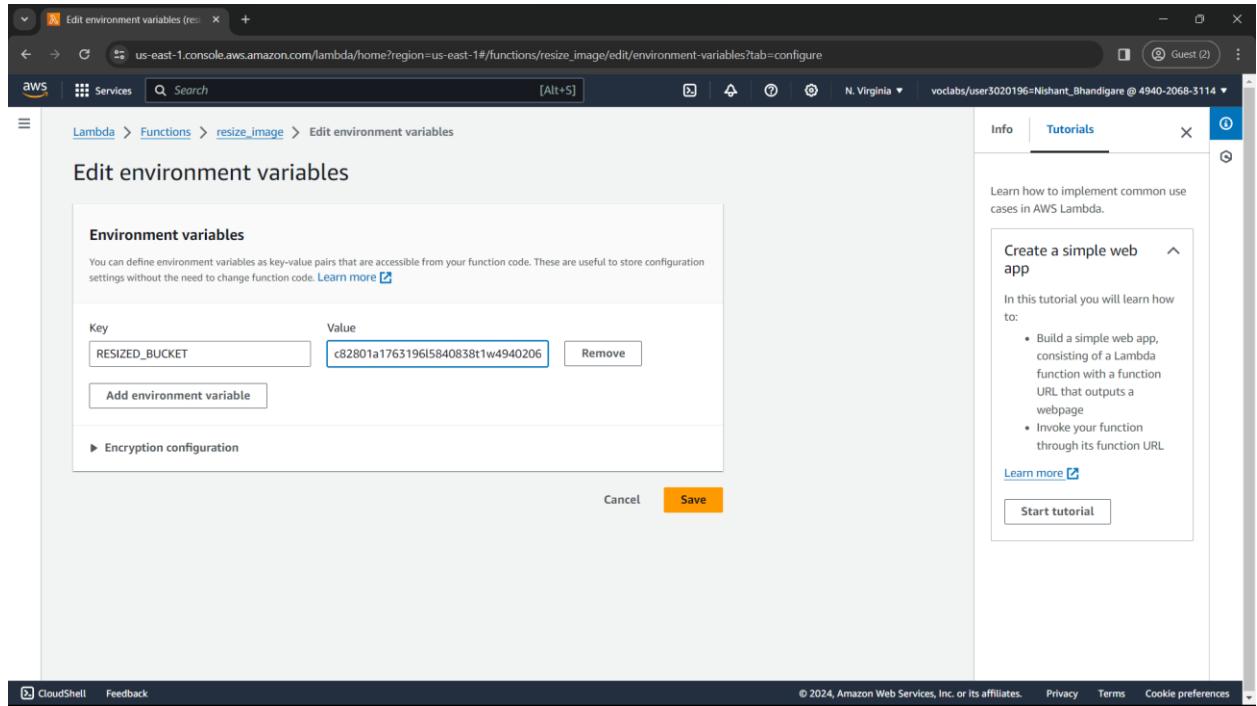
The screenshot shows the 'Edit environment variables' dialog box for the 'resize\_image' function. The title bar says 'Lambda | us-east-1'. The main area is titled 'Edit environment variables' and contains a 'Environment variables' section. It explains that environment variables are key-value pairs accessible from function code. A note says 'There are no environment variables on this function.' Below this is a 'Add environment variable' button. Further down is a 'Encryption configuration' section with a '▶' icon. At the bottom are 'Cancel' and 'Save' buttons. To the right of the dialog, there is a sidebar with 'Tutorials' for 'Create a simple web app' and a 'Start tutorial' button.

Enter the following values:

**Key:** RESIZED\_BUCKET

21. **Value:** Enter the value that you retrieved in step 17.

## 22. Choose Save.



You have created a Lambda function.

## Task 2: Configuring an Amazon S3 trigger to invoke a Lambda function

In this task, you configure an S3 trigger on an existing S3 bucket and your Lambda function. The Lambda function resizes images and places them in another bucket.

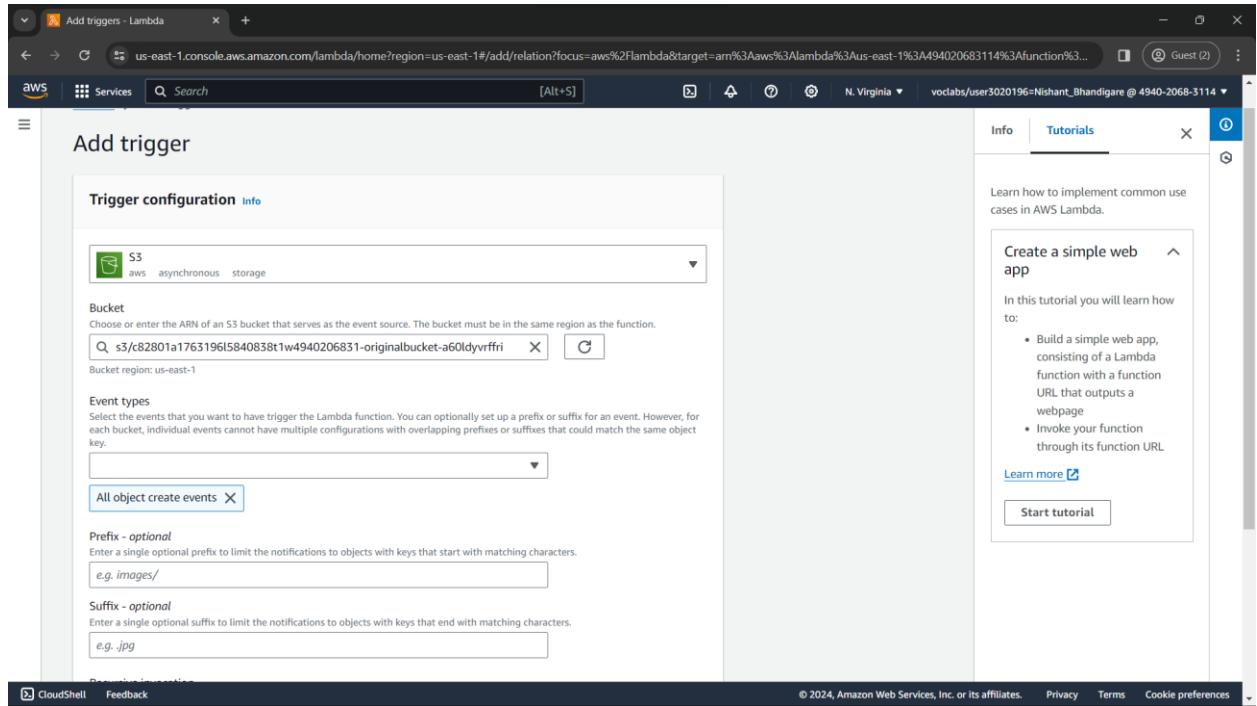
23. In the **Function overview** section of the Lambda console near the top of the page, choose **Add trigger**.

24. In the **Trigger configuration** section, choose **S3** from the dropdown list.

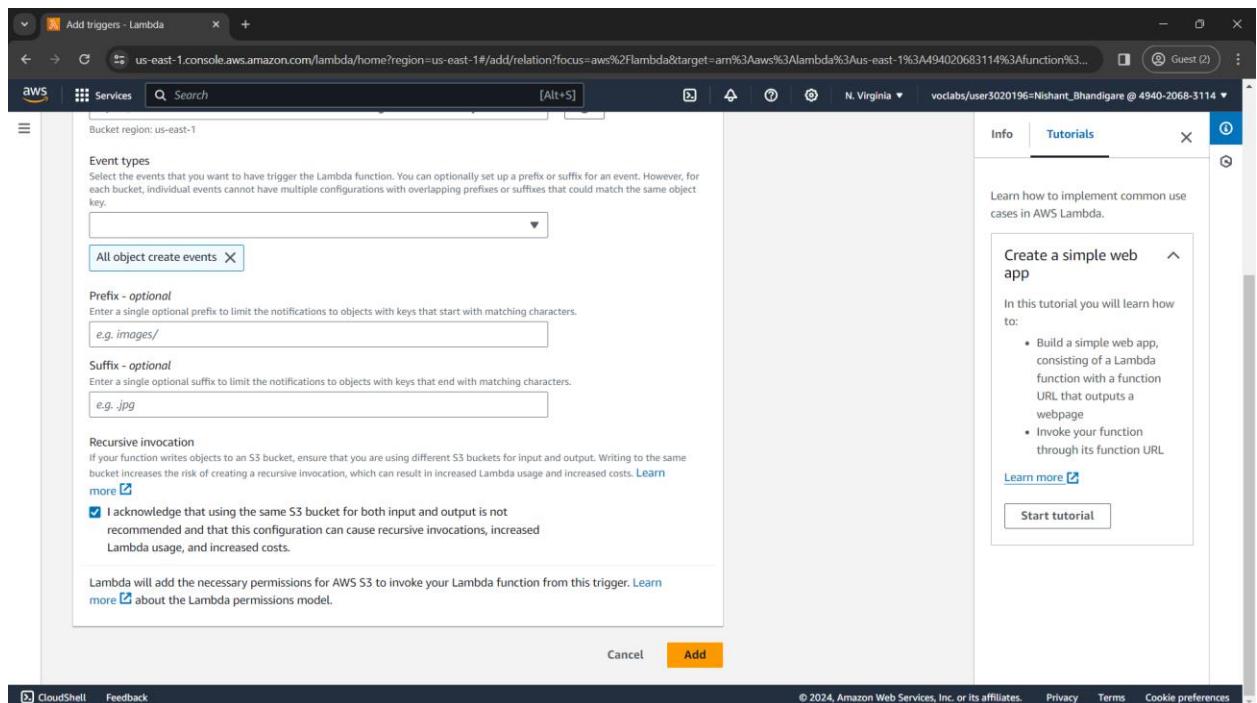
25. For Bucket, choose the bucket with **original** in the name.

26. For Event Type, choose **All object create events**.

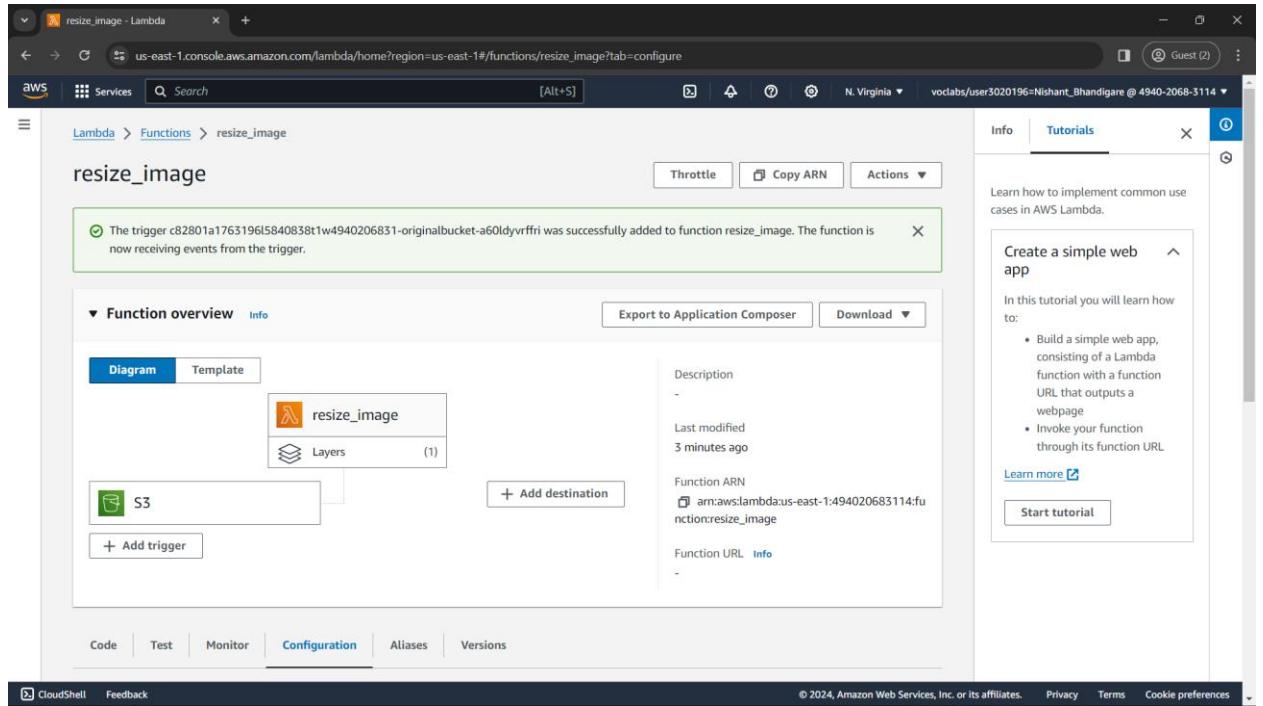
27. Acknowledge the notification for **Recursive invocation** by selecting the check box.



## 28. Choose Add.



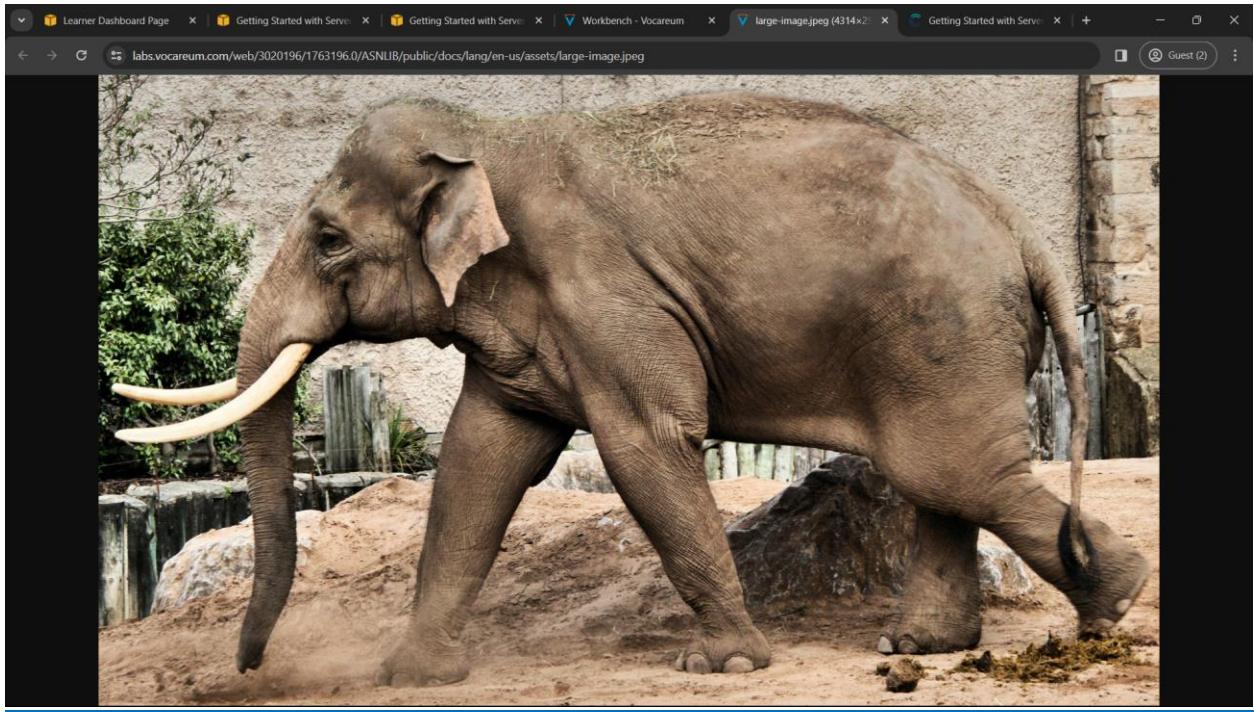
You have configured your Lambda function to be initiated when a new object is uploaded to the S3 bucket.



### Task 3: Uploading an image to the Amazon S3 bucket

In this task, you upload an image file to your bucket.

29. Open the context (right-click) menu for the following link, and download the file to your computer:
  - [large-image.jpg](#)



30. In the **AWS Management Console**, on the **Services** menu, enter **S3**. From the search results, choose **S3**.

A screenshot of the AWS Management Console. The search bar at the top contains 's3'. The left sidebar shows the Lambda service with a message: 'The trial period for Lambda has now ended.' Below it, under 'Functions', there are sections for 'Diagrams' and 'Logs'. The main search results page for 'Services' shows the 'S3' card, which is highlighted with a blue border. Other services listed include 'S3 Glacier', 'AWS Snow Family', and 'Storage Gateway'. Below the services, under 'Features', there are cards for 'Imports from S3' and 'Batch Operations'. To the right of the search results, a 'Tutorials' pane is open, showing a 'Create a simple web app' tutorial with a description and a 'Start tutorial' button.

31. Choose the link for the bucket that has **original** in the name.

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with options like Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings, Storage Lens (Dashboards, Storage Lens groups, AWS Organizations settings), Feature spotlight, and AWS Marketplace for S3. The main area is titled "Amazon S3" and "Account snapshot". It displays an "Account snapshot" card with a message about storage usage and activity trends, a "View Storage Lens dashboard" button, and tabs for "General purpose buckets" and "Directory buckets". Under "General purpose buckets", there's a table with four entries:

Name	AWS Region	Access	Creation date
c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrrffr	US East (N. Virginia) us-east-1	Bucket and objects not public	February 29, 2024, 14:27:07 (UTC+05:30)
c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrrffr	US East (N. Virginia) us-east-1	Bucket and objects not public	February 29, 2024, 14:27:07 (UTC+05:30)
c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrrffr	US East (N. Virginia) us-east-1	Bucket and objects not public	February 29, 2024, 14:27:08 (UTC+05:30)

At the bottom, there are buttons for CloudShell, Feedback, and links to Privacy, Terms, and Cookie preferences.

## 32. Choose Upload.

The screenshot shows the AWS S3 console for the bucket "c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrrffr". The sidebar is identical to the previous screenshot. The main area shows the bucket name and a breadcrumb trail: Amazon S3 > Buckets > c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrrffr. Below this, there are tabs for Objects, Properties, Permissions, Metrics, Management, and Access Points. The Objects tab is selected, showing a table with one entry: "No objects". A message states "You don't have any objects in this bucket." At the bottom, there's a "Upload" button. The footer includes CloudShell, Feedback, and links to Privacy, Terms, and Cookie preferences.

## 33. Choose Add files.

34. Choose the file that you downloaded.

The screenshot shows the AWS S3 'Upload objects' interface. At the top, the URL is <https://s3.console.aws.amazon.com/s3/upload/c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrfri?region=us-east-1&bucketType=general>. The page title is 'Upload objects - S3 bucket c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrfri'. The main area shows a file named 'large-image.jpeg' selected for upload. The destination is set to the same bucket. The 'Upload' button is located at the bottom right of the form.

Choose **Upload**.

35. Your file is uploaded to the bucket.

The screenshot shows the AWS S3 'Upload objects' interface after the file has been uploaded. The 'Files and folders' table now lists the uploaded file 'large-image.jpeg'. The 'Upload' button is highlighted in orange, indicating it is the active button.

36. Choose **Close**.

The screenshot shows the AWS S3 console with a green success message: "Upload succeeded". Below it, the "Upload: status" page displays a summary of the upload. The destination is listed as "s3://c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrfri". The status shows "Succeeded" with "1 file, 4.9 MB (100.00%)". There are also sections for "Files and folders" and "Configuration". A table lists the uploaded file: Name: large-image..., Folder: -, Type: image/jpeg, Size: 4.9 MB, Status: Succeeded.

If you are working on Task 4, you can now stop.

When your Lambda function runs correctly, the image file that you uploaded is reduced in size and placed in the S3 bucket that you specified when you set the environment variable for **RESIZED\_BUCKET**.

### 37. Return to the Buckets section in the S3 Console.

The screenshot shows the AWS S3 console with the "Buckets" section selected. It displays an "Account snapshot" and a table of "General purpose buckets". The table has columns: Name, AWS Region, Access, and Creation date. It lists three buckets: "c82801a1763196l5840838t1w4940206831-originalbucket-a60ldyvrfri" (Region: US East (N. Virginia) us-east-1, Access: Bucket and objects not public, Created: February 29, 2024, 14:27:07 (UTC+05:30)), "c82801a1763196l5840838t1w4940206831-resizedbucket-nmtz2b2aaqkg" (Region: US East (N. Virginia) us-east-1, Access: Bucket and objects not public, Created: February 29, 2024, 14:27:08 (UTC+05:30)), and "02068311-lambdalayerbucket-0kpiib5rzava" (Region: US East (N. Virginia) us-east-1, Access: Bucket and objects not public, Created: February 29, 2024, 14:27:07 (UTC+05:30)).

38. Choose the link for the bucket that has **resized** in the name.

Notice the file size. It's significantly reduced from the original size of 4.9 MB.

The screenshot shows the AWS S3 console interface. On the left, there is a navigation sidebar with sections like 'Buckets', 'Storage Lens', and 'AWS Marketplace for S3'. The main area displays a single object named 'large-image.jpeg' in a table format. The table includes columns for Name, Type, Last modified, Size, and Storage class. The object details show it was last modified on February 29, 2024, at 14:50:12 (UTC+05:30), is 573.7 KB in size, and is stored in the Standard storage class. There are also buttons for Actions, Create folder, and Upload.

Name	Type	Last modified	Size	Storage class
large-image.jpeg	jpeg	February 29, 2024, 14:50:12 (UTC+05:30)	573.7 KB	Standard

When you uploaded the image file, S3 initiated the **resize\_image** Lambda function. Review the logs to see how your function performed.

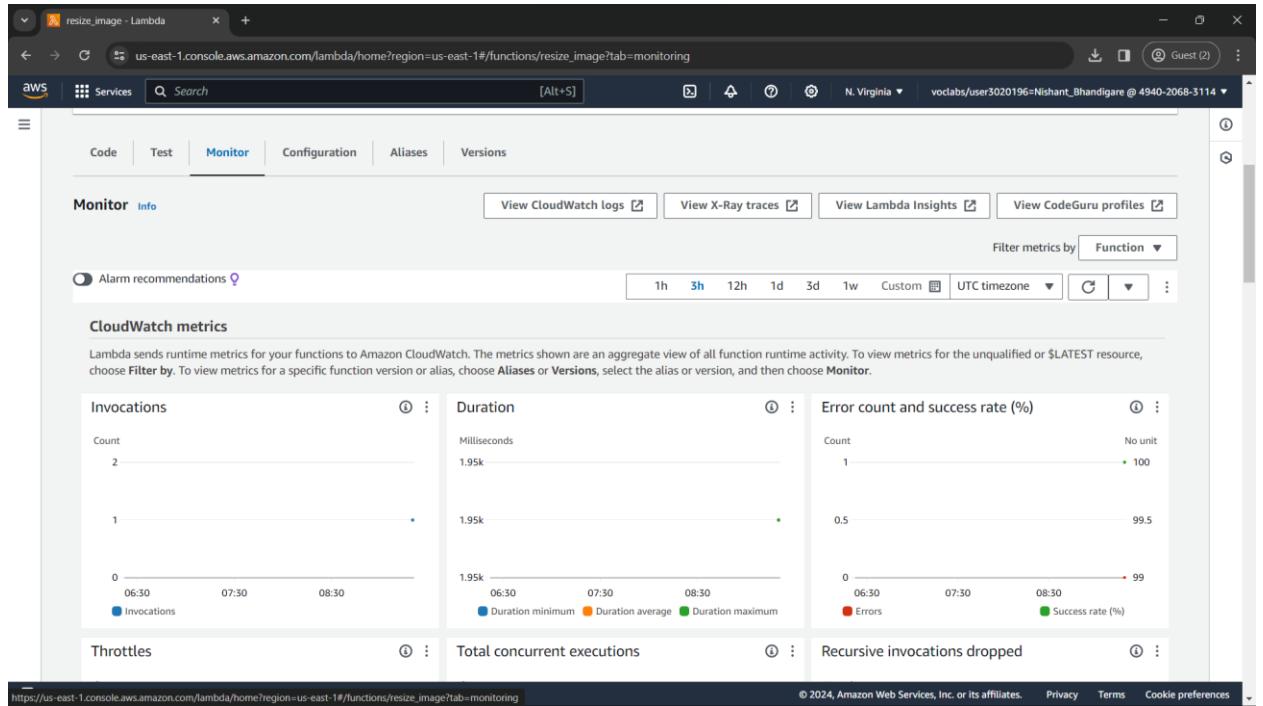
39. Navigate back to the Lambda console.

The screenshot shows the AWS Lambda Functions page. On the left, a sidebar menu includes 'Dashboard', 'Applications', 'Functions' (which is selected), 'Additional resources', and 'Related AWS resources'. The main area displays a table titled 'Functions (2)'. The table has columns for 'Function name', 'Description', 'Package type', 'Runtime', and 'Last modified'. It lists two entries: 's3Mover' (Lambda function to move files from one bucket to the other, Zip, Python 3.9, 24 minutes ago) and 'resize\_image' (Description: -, Zip, Python 3.9, 5 minutes ago). A search bar at the top of the table says 'Filter by tags and attributes or search by keyword'. At the bottom right of the table, there are 'Actions' and a 'Create function' button.

40. Choose the link for the **resize\_image** function.

The screenshot shows the AWS Lambda Function configuration page for 'resize\_image'. The top navigation bar shows the URL 'us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/resize\_image?tab=config'. The left sidebar shows 'Lambda > Functions > resize\_image'. The main area is titled 'resize\_image'. It features a 'Function overview' section with tabs for 'Diagram' (selected) and 'Template'. The diagram shows the function 'resize\_image' with a dependency on 'Layers' and an S3 trigger. There are buttons for '+ Add destination' and '+ Add trigger'. To the right, there are sections for 'Description', 'Last modified' (5 minutes ago), 'Function ARN' (arn:aws:lambda:us-east-1:494020683114:function:resize\_i mage), and 'Function URL' (Info). Below this is a 'Configuration' tab with sub-tabs for 'General configuration' (selected), 'Triggers (1)', and 'Aliases' and 'Versions'. The 'Triggers (1)' tab shows a single trigger for 'S3'. A footer bar at the bottom includes 'CloudShell', 'Feedback', and copyright information '© 2024, Amazon Web Services, Inc. or its affiliates.'.

41. Choose the **Monitor** tab.



## 42. Choose Logs.

In the **Recent Invocations** table, choose a row to expand the details.

43. Notice the metrics that are recorded for each function invocation. You might have to wait for up to one minute for the data to be updated.

The **DurationInMS** column tells you how long your function ran for this invocation.

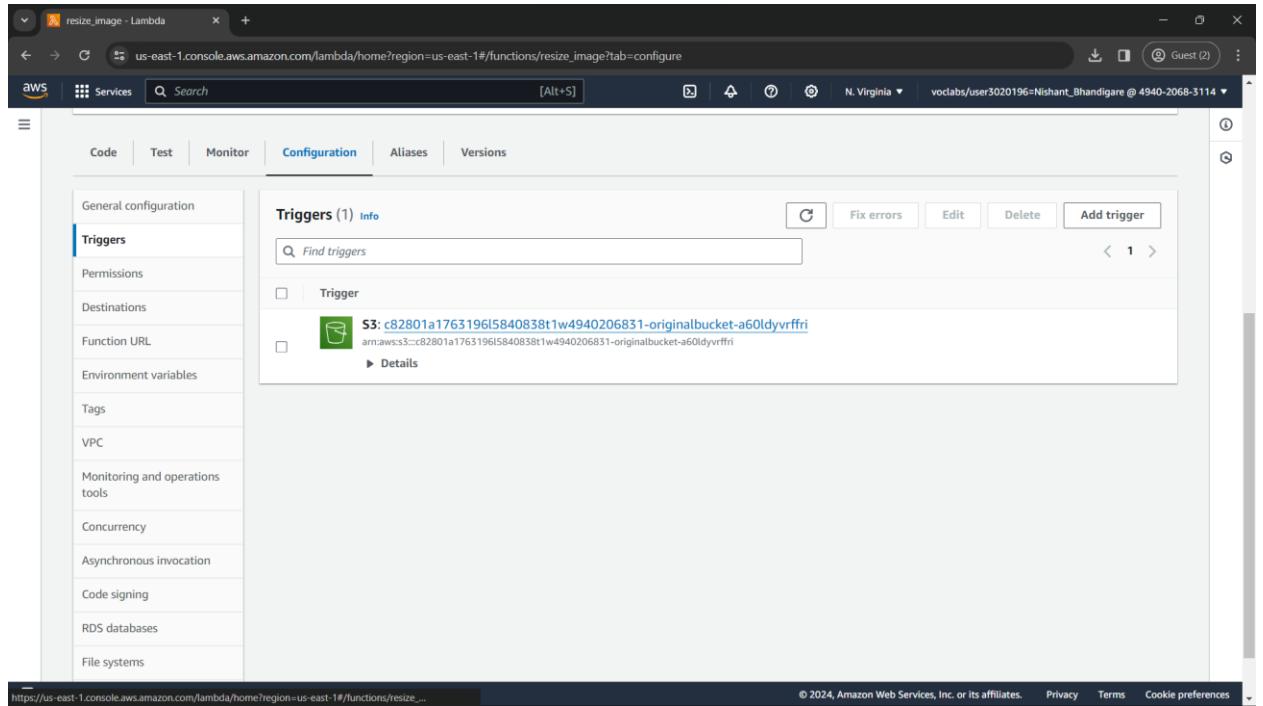
The first time that your Lambda function is invoked, the Lambda execution environment has to download your code and start a new execution environment. This process is called a cold start. The **@initDuration** metric in the **Recent invocation** details signifies the cold start time.

Note the **MemorySetInMB** column. The amount of memory that's available to your Lambda function can be adjusted to affect the performance of your Lambda function. The amount of memory also determines the amount of virtual CPU available to a function. Adding more memory proportionally increases the amount of CPU, which increases the overall computational power available. If a function is CPU-, network- or memory-bound, then changing the memory setting can dramatically improve its performance.

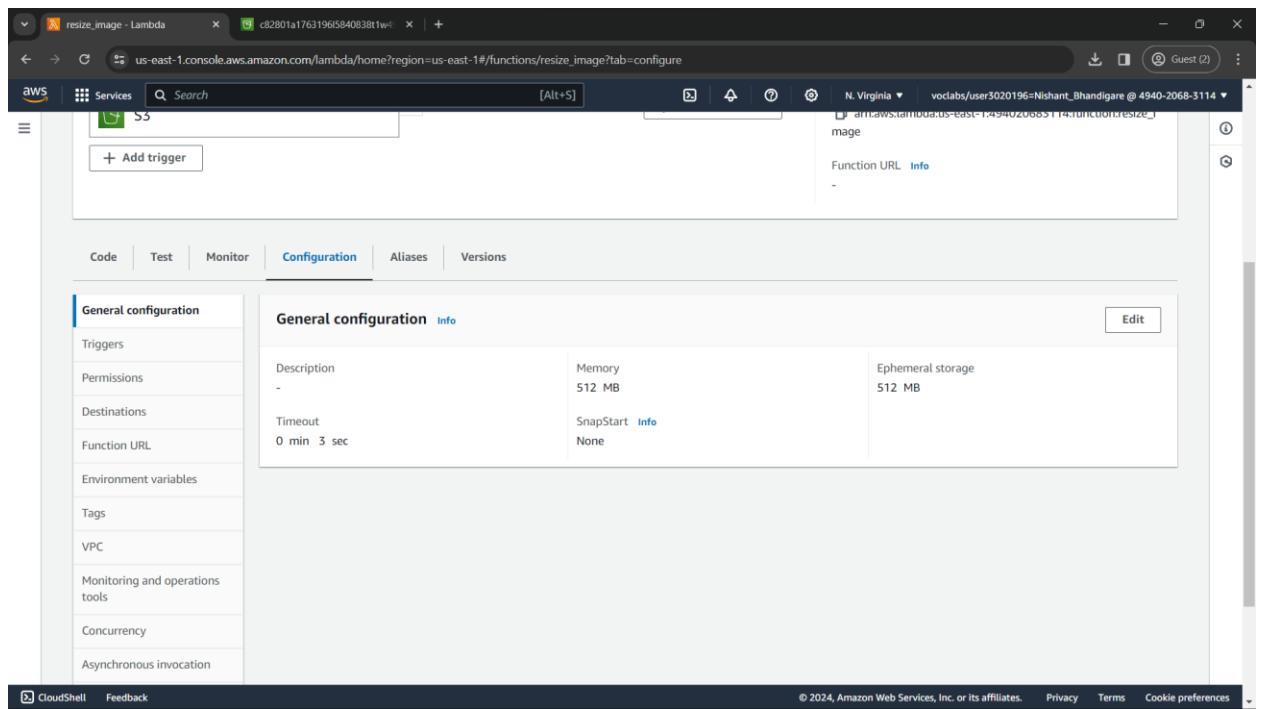
Notice how long it took your function to run. It could be faster. Adjust your Lambda function so that it runs faster.

## Task 4: Optimizing Lambda function memory for performance

44. Choose the **Configuration** tab.



#### 45. Choose General Configuration.



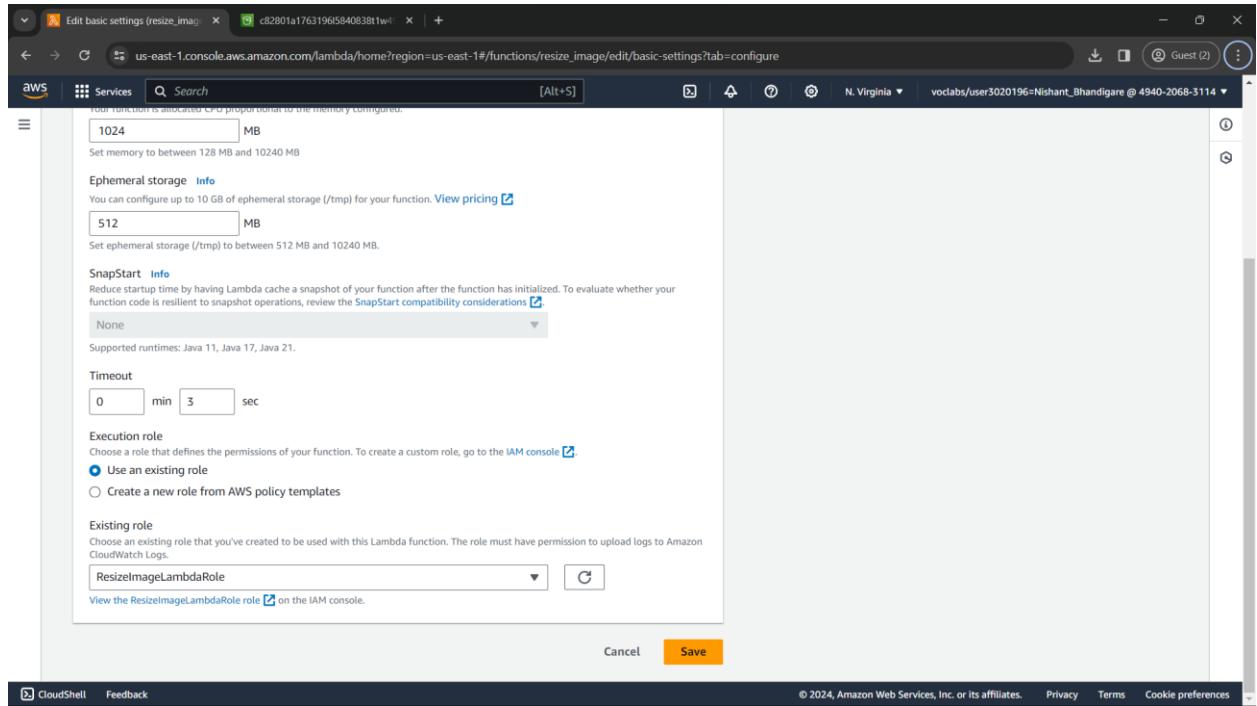
#### 46. Choose Edit.

The screenshot shows the AWS Lambda 'Edit basic settings' page for a function named 'resize\_image'. The 'Memory' field is currently set to 512 MB. The 'Timeout' field is set to 0 min and 3 sec. Other settings like 'Description', 'Ephemeral storage', and 'SnapStart' are also visible.

#### 47. Adjust Memory to 1024 MB.

The screenshot shows the AWS Lambda 'Edit basic settings' page for the same 'resize\_image' function. The 'Memory' field has been updated to 1024 MB. The 'Timeout' field remains at 0 min and 3 sec. The rest of the configuration is identical to the previous screenshot.

#### 48. Choose Save.



49. Upload your image to the original S3 bucket again. If you need help, review steps [31-37](#).

50. Repeat steps 45-50 using 2048 MB and 3008 MB as the value for **Memory**.

Your function now runs in approximately 500 milliseconds with 3008 MB of memory and an image that is 5 MB.

**Task 4: Optimizing Lambda function memory for performance**

48. Choose the **Configuration** tab.  
 49. Choose **General Configuration**.  
 50. Choose **Edit**.  
 51. Adjust **Memory** to 1024 MB.  
 52. Choose **Save**.  
 53. Upload your image to the original S3 bucket again. If you need help, review steps [35-41](#).  
 54. Repeat steps 49-54 using 2048 MB and 3008 MB as the value for **Memory**.

Your function now runs in approximately 500 milliseconds with 3008 MB of memory and an image that is 5 MB.

**Summary**

In this lab, you created a Lambda function and configured an S3 trigger on that Lambda function. You uploaded an image to an S3 bucket which initiated the Lambda function to resize the image and place it in another bucket. After reviewing the logs to see performance metrics, you optimized the Lambda function by increasing the available memory for the function.

Excellent work!

**Lab complete**

**Congratulations!** You have completed the lab.

55. Choose **End Lab** at the top of this page, and then select **Yes** to confirm that you want to end the lab. A panel indicates that **DELETE** has been initiated... You may close this message box now.

56. Choose the X in the top right corner to close the panel.

©2023 Amazon Web Services, Inc. and its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.

Successfully updated the function **resize\_image**.

14 seconds ago

Function ARN  
arn:aws:lambda:us-east-1:8362246467:function:resize\_image

Function URL [Info](#)

Lambda (1)

S3

+ Add destination

+ Add trigger

Code Test Monitor Configuration Aliases Versions

General configuration	
Description	Memory
512 MB	3008 MB
Ephemeral storage	Timeout
None	0 min 3 sec
Tags	SnapStart
VPC	Info
Monitoring and observability	None

**Conclusion:** In conclusion, the journey to study and implement Platform as a Service (PaaS) using AWS or Microsoft Azure has provided valuable insights into the capabilities and advantages of cloud-native application development. Throughout this endeavor, we have delved into the foundational principles of PaaS offerings provided by AWS and Azure, understanding their role in simplifying application development, deployment, and management. By leveraging services like AWS Elastic Beanstalk, AWS Lambda, Azure App Service, and Azure Functions, we have experienced firsthand the agility and scalability offered by PaaS solutions, enabling rapid iteration and innovation in application development. Furthermore, the experiment has underscored the importance of selecting the appropriate PaaS services based on application requirements, as well as the significance of seamless integration with other cloud services for enhanced functionality. Overall, the study and implementation of PaaS using AWS or Microsoft Azure have equipped us with the knowledge and skills to streamline the application development lifecycle, accelerate time-to-market, and drive business growth in the cloud era.