Hands-On Lab: Event-Driven Thumbnail Generation

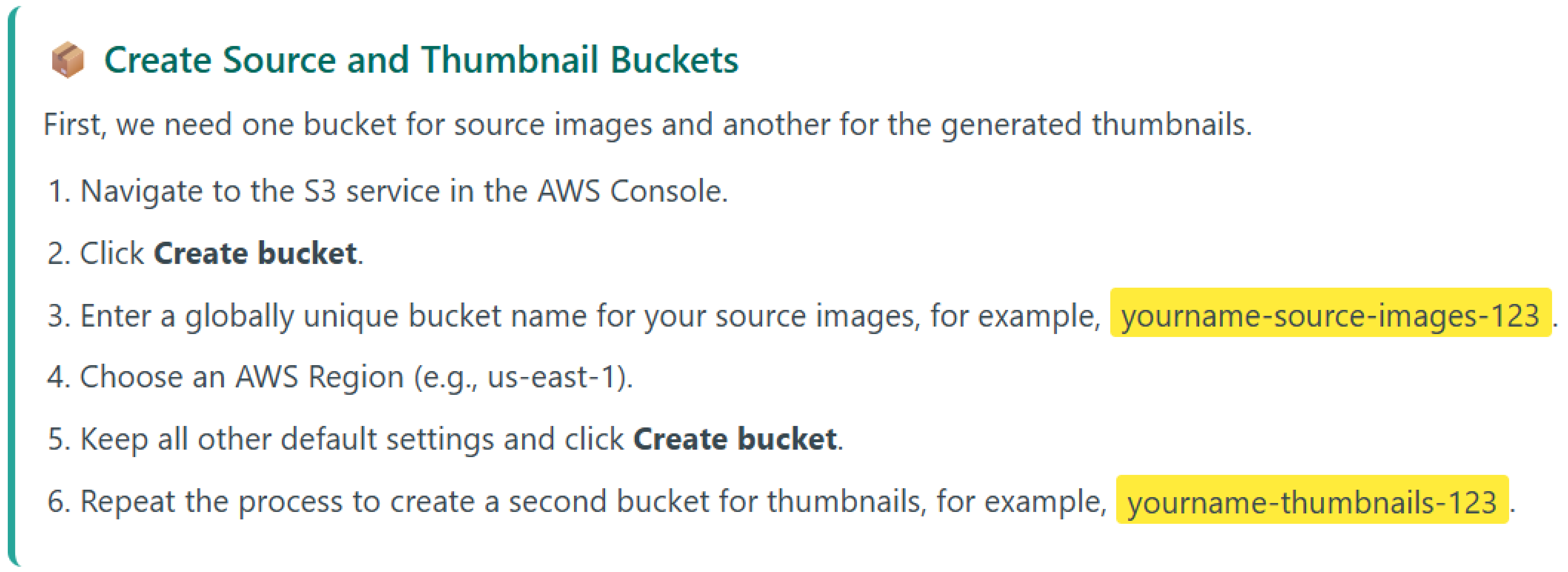
🎯 Objective

To build a simple pipeline where uploading an image to an S3 bucket automatically triggers a Lambda function to create a thumbnail and save it in another S3 bucket. This demonstrates a real-world, event-driven use case for AWS Lambda.

📋 Prerequisites

- An AWS Account with administrative access
- Access to the AWS Management Console

Lab Step 1: Create Two S3 Buckets

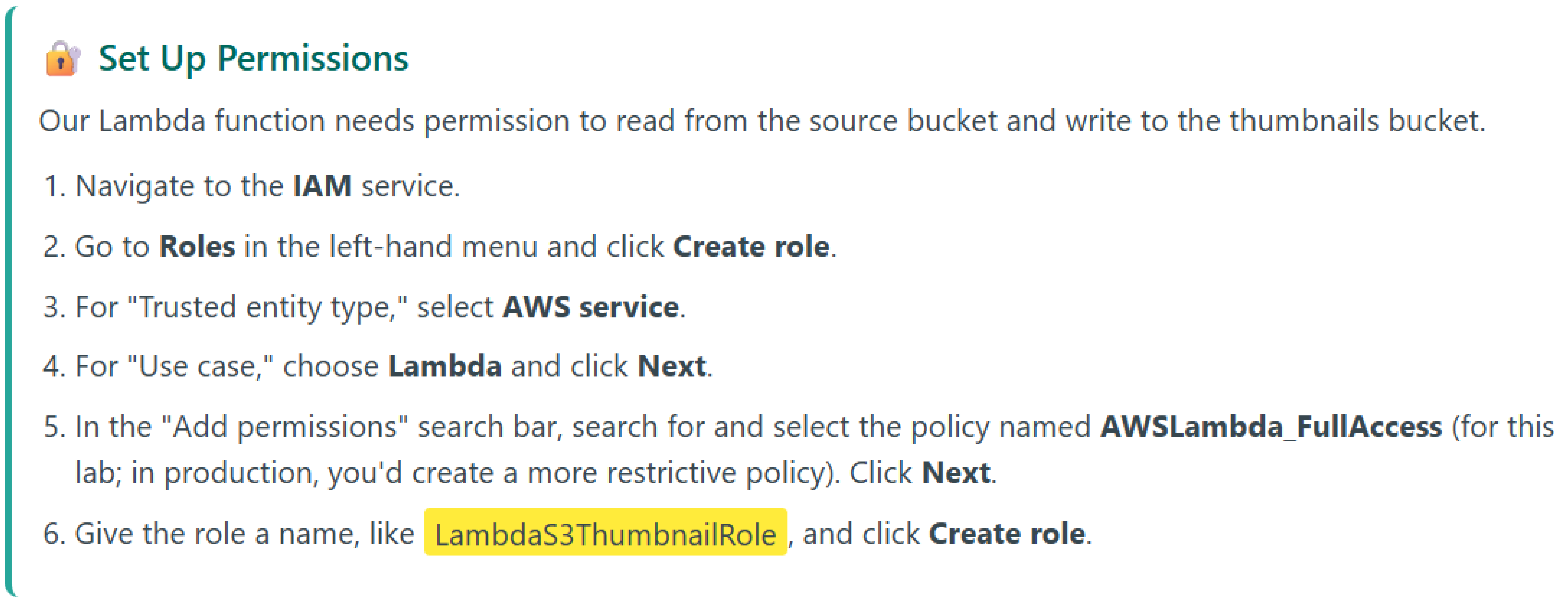


📦 Create Source and Thumbnail Buckets

First, we need one bucket for source images and another for the generated thumbnails.

1. Navigate to the S3 service in the AWS Console.
2. Click **Create bucket**.
3. Enter a globally unique bucket name for your source images, for example, `yourname-source-images-123`.
4. Choose an AWS Region (e.g., us-east-1).
5. Keep all other default settings and click **Create bucket**.
6. Repeat the process to create a second bucket for thumbnails, for example, `yourname-thumbnails-123`.

Lab Step 2: Create an IAM Role for Lambda



🔐 Set Up Permissions

Our Lambda function needs permission to read from the source bucket and write to the thumbnails bucket.

1. Navigate to the **IAM** service.
2. Go to **Roles** in the left-hand menu and click **Create role**.
3. For "Trusted entity type," select **AWS service**.
4. For "Use case," choose **Lambda** and click **Next**.
5. In the "Add permissions" search bar, search for and select the policy named **AWSLambda_FullAccess** (for this lab; in production, you'd create a more restrictive policy). Click **Next**.
6. Give the role a name, like `LambdaS3ThumbnailRole`, and click **Create role**.

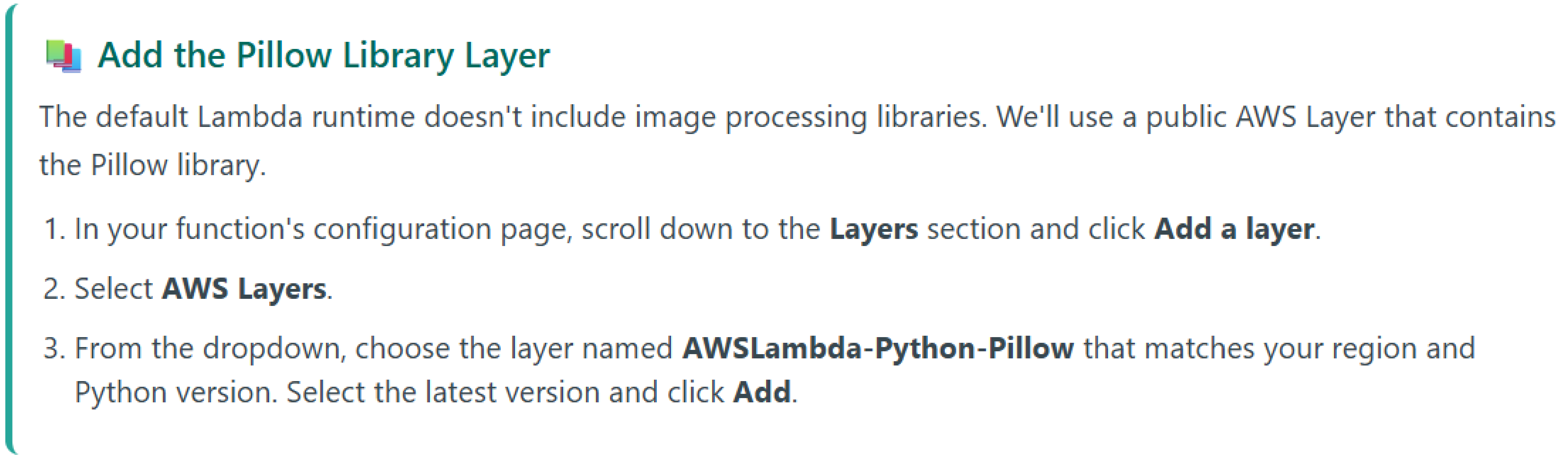
Lab Step 3: Create the Lambda Function

Build Your Function

Now we'll create the function that performs the image resizing.

1. Navigate to the **Lambda** service in the AWS Console.
2. Click **Create function**.
3. Select **Author from scratch**.
4. Enter a Function name, like `ThumbnailGenerator`.
5. For Runtime, select **Python 3.9** (or a newer Python version).
6. For Architecture, keep **x86_64**.
7. Under Permissions, expand "Change default execution role."
8. Select **Use an existing role** and choose the **LambdaS3ThumbnailRole** you created in the previous step.
9. Click **Create function**.

Lab Step 4: Add Code and Helper Layer (Part 1)



📚 Add the Pillow Library Layer

The default Lambda runtime doesn't include image processing libraries. We'll use a public AWS Layer that contains the Pillow library.

1. In your function's configuration page, scroll down to the **Layers** section and click **Add a layer**.
2. Select **AWS Layers**.
3. From the dropdown, choose the layer named **AWSLambda-Python-Pillow** that matches your region and Python version. Select the latest version and click **Add**.

Lab Step 4: Add Code and Helper Layer (Part 2)

💻 Add the Python Code

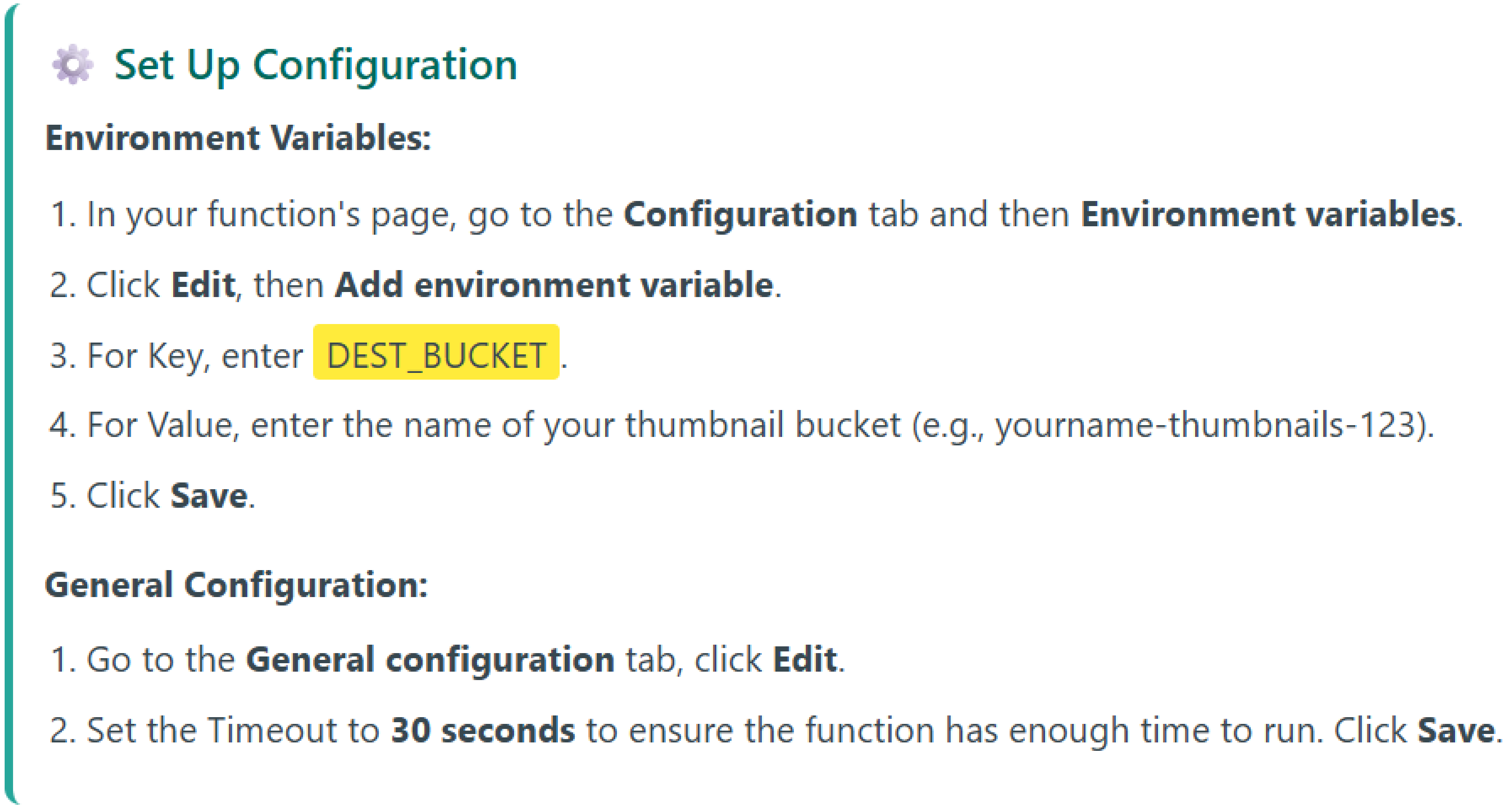
Now, go back to the **Code source** section. Replace the existing code in **lambda_function.py** with the following:

```
import boto3 import os from urllib.parse import unquote_plus from PIL import Image import uuid s3_client =
boto3.client('s3') def resize_image(image_path, resized_path): with Image.open(image_path) as image:
image.thumbnail((128, 128)) image.save(resized_path) def lambda_handler(event, context): for record in
event['Records']: source_bucket = record['s3']['bucket']['name'] key = unquote_plus(record['s3']['object']
['key']) dest_bucket = os.environ['DEST_BUCKET'] download_path = f'/tmp/{uuid.uuid4()}{key}' upload_path =
f'/tmp/resized-{key}' s3_client.download_file(source_bucket, key, download_path)
resize_image(download_path, upload_path) s3_client.upload_file(upload_path, dest_bucket, f'resized-{key}')
return {'statusCode': 200, 'body': 'Thumbnail generation successful!'}
```

Click the **Deploy** button to save your code.

lambda_example.txt

Lab Step 5: Configure Environment Variables



Set Up Configuration

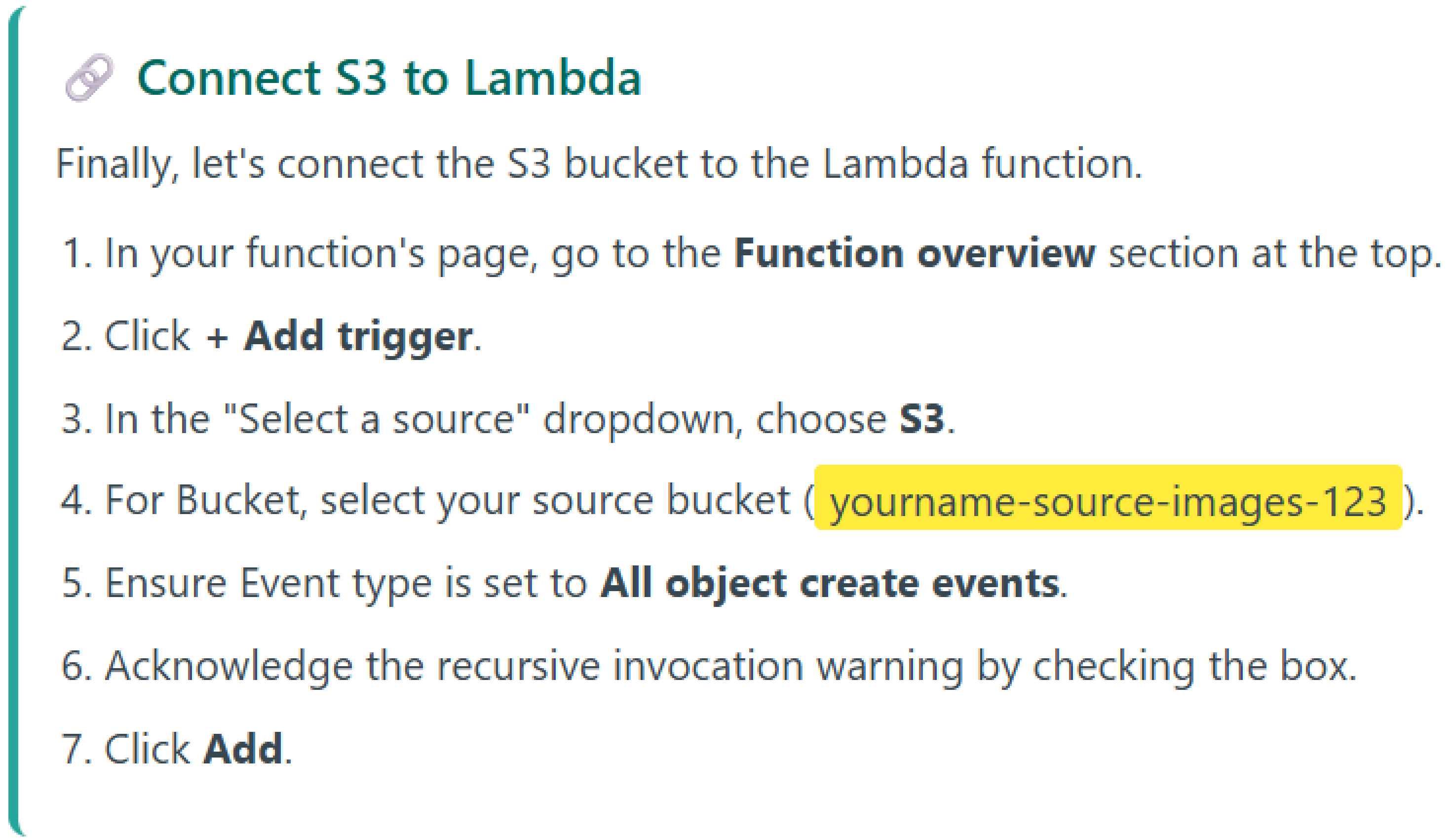
Environment Variables:

1. In your function's page, go to the **Configuration** tab and then **Environment variables**.
2. Click **Edit**, then **Add environment variable**.
3. For Key, enter `DEST_BUCKET`.
4. For Value, enter the name of your thumbnail bucket (e.g., yourname-thumbnails-123).
5. Click **Save**.

General Configuration:

1. Go to the **General configuration** tab, click **Edit**.
2. Set the Timeout to **30 seconds** to ensure the function has enough time to run. Click **Save**.

Lab Step 6: Add the S3 Trigger



🔗 Connect S3 to Lambda

Finally, let's connect the S3 bucket to the Lambda function.

1. In your function's page, go to the **Function overview** section at the top.
2. Click **+ Add trigger**.
3. In the "Select a source" dropdown, choose **S3**.
4. For Bucket, select your source bucket (`yourname-source-images-123`).
5. Ensure Event type is set to **All object create events**.
6. Acknowledge the recursive invocation warning by checking the box.
7. Click **Add**.

Lab Step 7: Test the Pipeline!

🎉 See Your Serverless Solution in Action

1. Navigate back to the **S3 service**.
2. Go into your source bucket (yourname-source-images-123).
3. Click **Upload** and add any **.jpg** or **.png** image from your computer.
4. Once the upload is complete, navigate to your thumbnails bucket (yourname-thumbnails-123).
5. After a few seconds, you should see a new file appear named **resized-your-image-name.jpg**.
6. Download and open it - it will be a 128x128 thumbnail of your original image!

🎊 Congratulations!

You have successfully built an event-driven serverless pipeline using AWS Lambda and S3.

Let's Get Building!

Hands-On Lab Time

Please follow the lab guide carefully.

Don't hesitate to ask questions!

Lab Step 1: Create Two S3 Buckets

First, we need one bucket for source images and another for the generated thumbnails.

- Navigate to the S3 service in the AWS Console
- Click **Create bucket**
- Enter a globally unique bucket name for your source images (e.g., *yourname-source-images-123*)
- Choose an AWS Region (e.g., us-east-1)
- Keep all other default settings and click **Create bucket**
- Repeat the process to create a second bucket for thumbnails (e.g., *yourname-thumbnails-123*)

🛡️ Lab Step 2: Create an IAM Role for Lambda

Our Lambda function needs permission to read from the source bucket and write to the destination bucket.

- Navigate to the IAM service
- Go to **Roles** in the left menu and click **Create role**
- For "Trusted entity type," select **AWS service**
- For "Use case," choose **Lambda** and click **Next**

🛡️ Lab Step 2: Create an IAM Role (continued)

- On the "Add permissions" page, search for and add the following two policies:
  - **AWSLambdaBasicExecutionRole** (for logging)
  - **AmazonS3FullAccess** (For this lab. In production, use a more restrictive custom policy)
- Click **Next**
- Give the role a name, like *LambdaS3ThumbnailRole*
- Click **Create role**

📦 Lab Step 3: Prepare the Lambda Deployment Package

This is the crucial step! We will prepare our Python code and its dependencies on our local machine.

This is the modern and more reliable approach - packaging the necessary library (Pillow) directly with your Lambda function code in a .zip file.

This method gives you full control over your dependencies.

📦 Lab Step 3: Setup (Part 1)

1. Create a Project Folder:

On your computer, create a new folder named **thumbnail-project**

2. Create the Code File:

Inside the thumbnail-project folder, create a new file and name it **lambda_function.py**

3. Add the Python Code:

Open lambda_function.py and paste the Python code (shown on next slide)

Python Code for lambda_function.py

lambda_for_thumbnail.txt

📦 Lab Step 3: Install the Library

4. Install the Pillow Library:

- Open your computer's terminal or command prompt
- Navigate into the thumbnail-project folder using the cd command:
  `cd path/to/thumbnail-project`
- Run the following command to install Pillow inside your project folder:
  pip install --platform manylinux2014_x86_64 --target . --implementation cp --python-version 3.12 --only-binary=:all: Pillow
- The **-t .** flag tells pip to install the library in the current directory (.) instead of a central system location

📦 Lab Step 3: Create the ZIP File

5. Create the ZIP File:

- Open the thumbnail-project folder. You should see lambda_function.py along with several new folders (PIL, Pillow, etc.)
- Select **all the contents inside the folder**, right-click, and compress them into a .zip file
- **Important:** Do NOT zip the parent thumbnail-project folder itself. You must zip the files and folders that are inside it
- Name the file **thumbnail-lambda.zip**

⚙️ Lab Step 4: Create Lambda Function

Now we'll create the function in AWS and upload our code package.

- Navigate to the Lambda service and click **Create function**
- Select **Author from scratch**
- Function name: **ThumbnailGenerator**
- Runtime: Select **Python 3.12** (or newer Python version)
- Architecture: Keep **x86_64**
- Permissions: Expand "Change default execution role," select **Use an existing role**, and choose the **LambdaS3ThumbnailRole** you created
- Click **Create function**

⚙️ Lab Step 4: Upload & Configure

Upload the Code:

- In the "Code source" section, click the **Upload from** button
- Select **.zip file**, and upload the **thumbnail-lambda.zip** file
- Click **Save**

- **Configure Environment Variables:**
- Go to the **Configuration** tab and select **Environment variables**
- Click **Edit**, then **Add environment variable**
- Key: **DEST_BUCKET**
- Value: Enter the name of your thumbnails bucket (e.g., yourname-thumbnails-123)
- Click **Save**

⚙️ Lab Step 4: Configure Timeout

Configure Timeout:

- Still in the **Configuration** tab, select **General configuration**
- Click **Edit**
- Set the **Timeout** to **30 seconds**
- Click **Save**

Why 30 seconds? Image processing can take time, especially for larger files. The default 3-second timeout is too short for this task.

⚡ Lab Step 5: Add the S3 Trigger

Finally, let's connect the S3 bucket to trigger the Lambda function.

- In your function's page, go to the **Function overview** section at the top
- Click **+ Add trigger**
- In the "Select a source" dropdown, choose **S3**
- Bucket: Select your source bucket (e.g., yourname-source-images-123)
- Ensure **Event type** is set to **All object create events**
- Acknowledge the recursive invocation warning by checking the box
- Click **Add**

✅ Lab Step 6: Test the Pipeline!

Time to see it in action!

- Navigate back to the S3 service
- Go into your source bucket (yourname-source-images-123)
- Click **Upload** and add any .jpg or .png image from your computer
- Once the upload is complete, navigate to your thumbnails bucket (yourname-thumbnails-123)
- After a few seconds, you should see a new file appear named **resized-your-image-name.jpg**
- Download and open it—it will be a 128x128 thumbnail of your original image!

🎉 Congratulations! You've built a serverless pipeline! 🎉

Questions and Answers (Part 1)

Q: What does "serverless" mean in the context of AWS Lambda?

A: It means you can run code without provisioning or managing any servers. AWS handles all the underlying infrastructure, and you only focus on your code.

Q: What is an "event" in our lab?

A: The event is the action of uploading a new image file to our source S3 bucket. This event is what "triggers" our Lambda function to run.

Q: Why did we need to create an IAM Role?

A: For security. The IAM Role grants our Lambda function the specific permissions it needs to access other AWS services—in this case, to read an object from one S3 bucket and write a new object to another S3 bucket.

Questions and Answers (Part 2)

Q: In Step 3, why did we run pip install Pillow -t .? What does the -t . part do?

A: We ran this command to download the Pillow library, which is needed for image processing. The -t . flag tells pip to install the library in the current directory (.) instead of a central system location, so we can package it with our code in the .zip file.

Q: What is the purpose of the "environment variable" we configured in Lambda?

A: It allows us to pass configuration data to our code without hardcoding it. In our lab, we used it to tell our function the name of the destination S3 bucket, making the code more reusable.

Q: What would happen if we uploaded a text file (.txt) to our source S3 bucket?

A: The S3 trigger would still fire, and the Lambda function would run. However, the function would likely fail and log an error because the Pillow library would not be able to open a text file as an image. This shows the importance of error handling in real-world applications.