Process Amazon S3 Events with AWS Lambda

S3 Events overview

AWS Lambda is only one of three possible S3 Events Destinations, which also include SNS topics and SQS queues.

A direct connection to Lambda is ideal when you require simple compute tasks to be executed on-demand. Alternatively, you may configure S3 to simply publish a new message on an SNS topic and use this single event to trigger multiple destinations, such as mobile push, SMS messages, HTTP endpoints and SQS queues.

During this lab, you will see a simple scenario containing a single Lambda function invoked every time a new S3 Object is uploaded. Please note that the used model is non-stream based (i.e. async).

What type of events can be triggered?

There are only three main types of events, with a few sub-types depending on the API call used.

The three main event types are:

- **ObjectCreated** - Both creation and update of an S3 Object (Put, Post, Copy or CompleteMultipartUpload)
- **ObjectRemoved** - Deletion, batch-deletion or versioned object marked for deletion (Delete or DeleteMarkerCreated)
- **ReducedRedundancyLostObject** - RRS storage class object loss

Note that the events configuration must be setup on the S3 Buckets and that you won't receive notifications for failed operations.

In the next steps, you will create a new S3 Bucket and connect its ObjectCreated events to AWS Lambda.

Step 1 Create an S3 bucket

Introduction

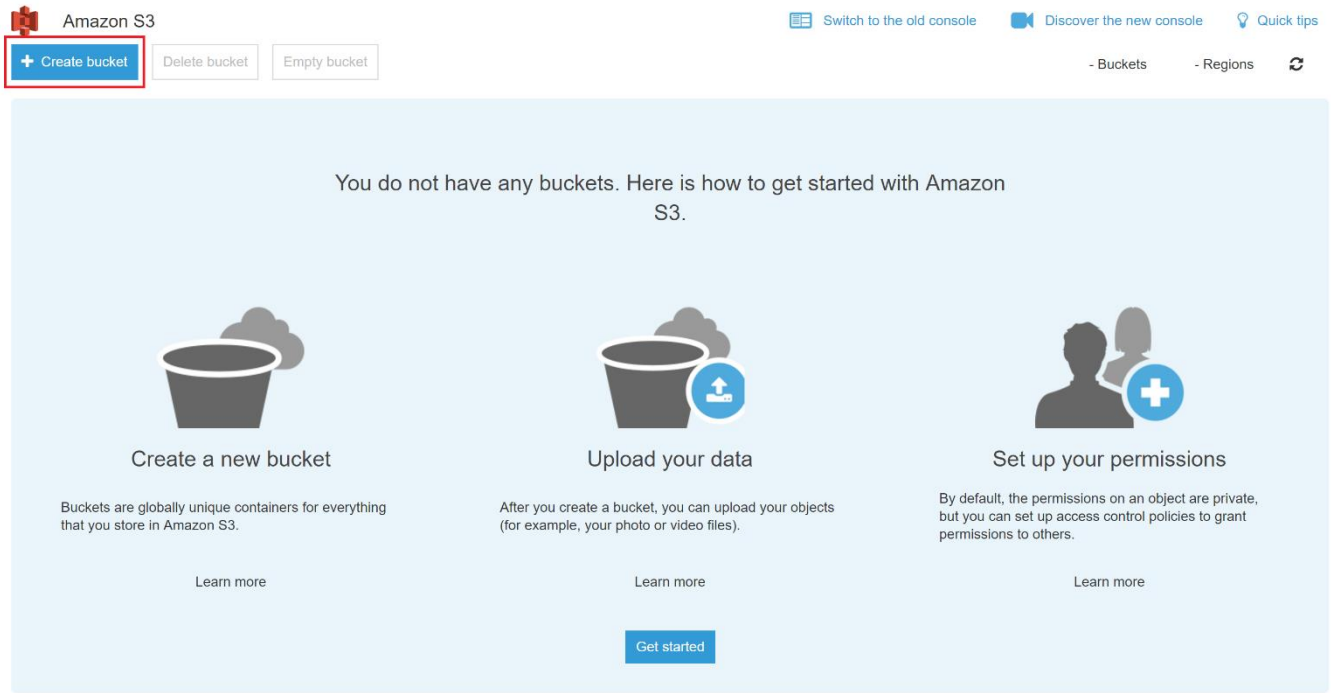
You can create an S3 bucket using the S3 management console. As with many other AWS services, you can also use the AWS API or CLI (command line interface) as well. This lab uses the AWS console however.

Instructions

1. Select the S3 service from the AWS Management Console under the **Storage** section:

You will be placed in the S3 console.

2. From the S3 console, click the blue **Create Bucket** button:



The **Create bucket** dialog box appears.

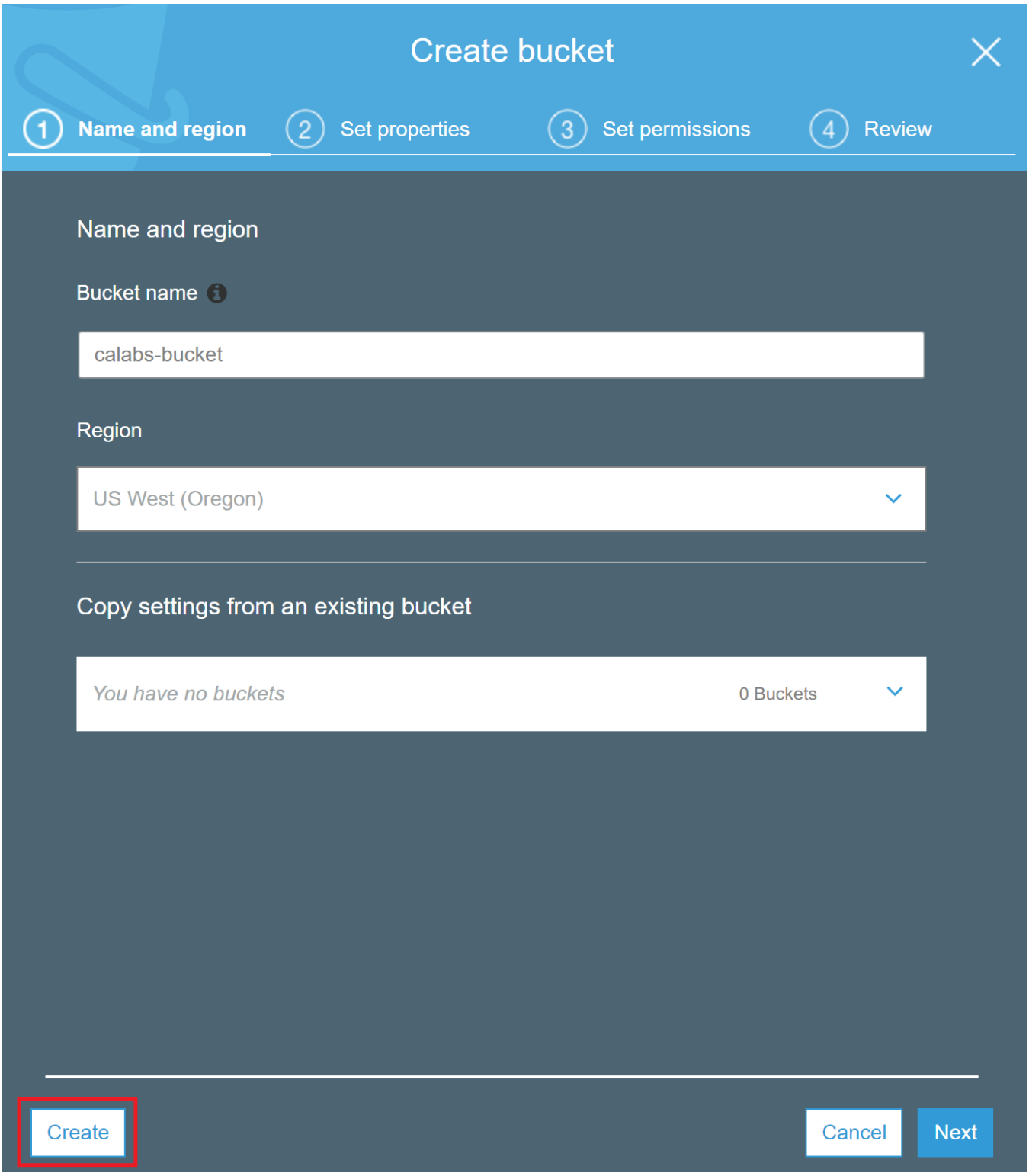
3. Next you will have to enter the **Bucket name** and select a **Region** from the selection box.

Bucket names must be globally unique, regardless of the AWS region in which you create the bucket, and they must be DNS-compliant.

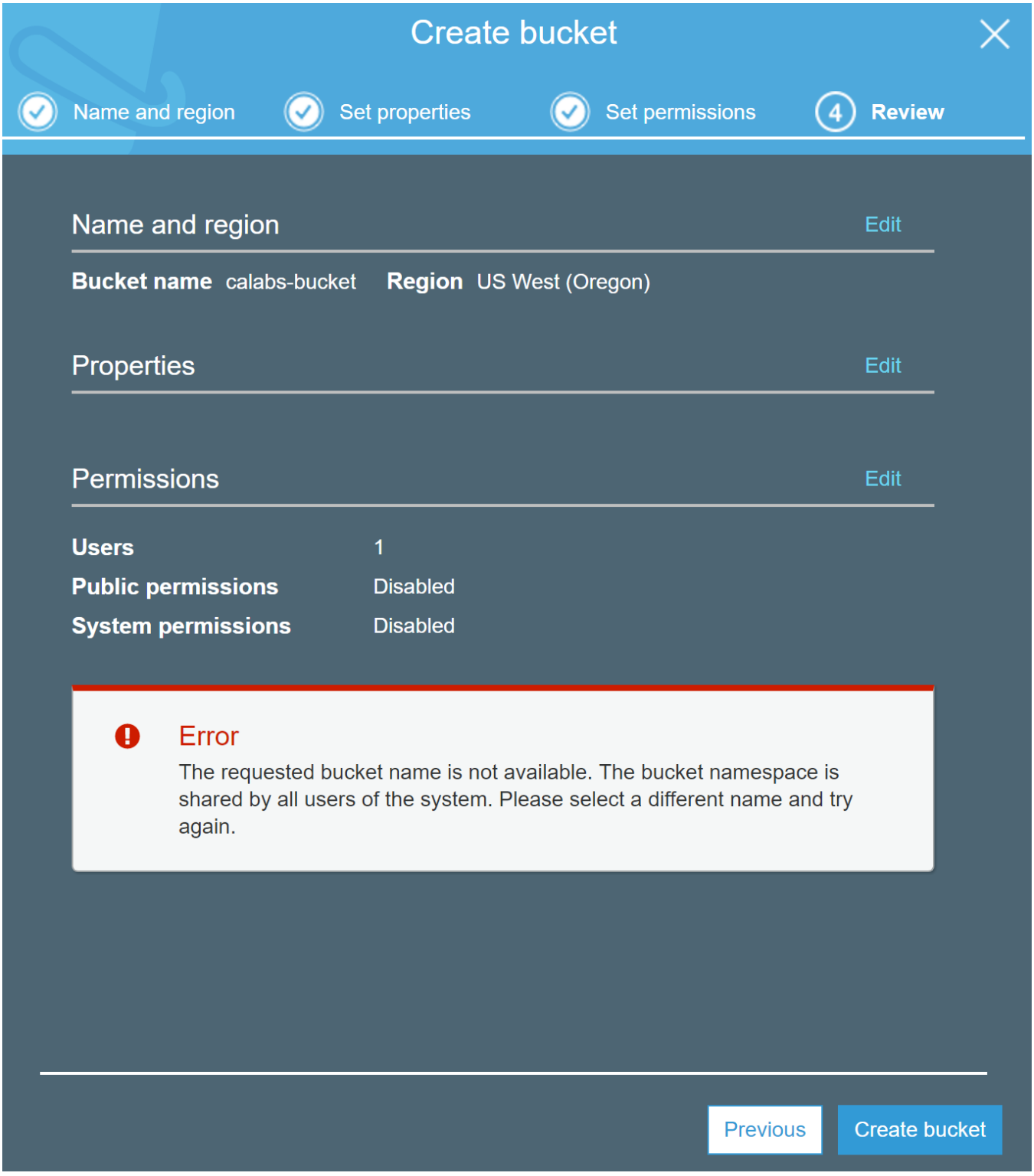
The rules for DNS-compliant bucket names are:

- Bucket names must be at least 3 and no more than 63 characters long.
- Bucket names can contain lowercase letters, numbers, periods, and/or hyphens. Each label must start and end with a lowercase letter or a number.
- Bucket names must not be formatted as an IP address (e.g., 192.168.1.1).

Please enter the following bucket name: `calabs-bucket-XXXX` and then click **Create:**

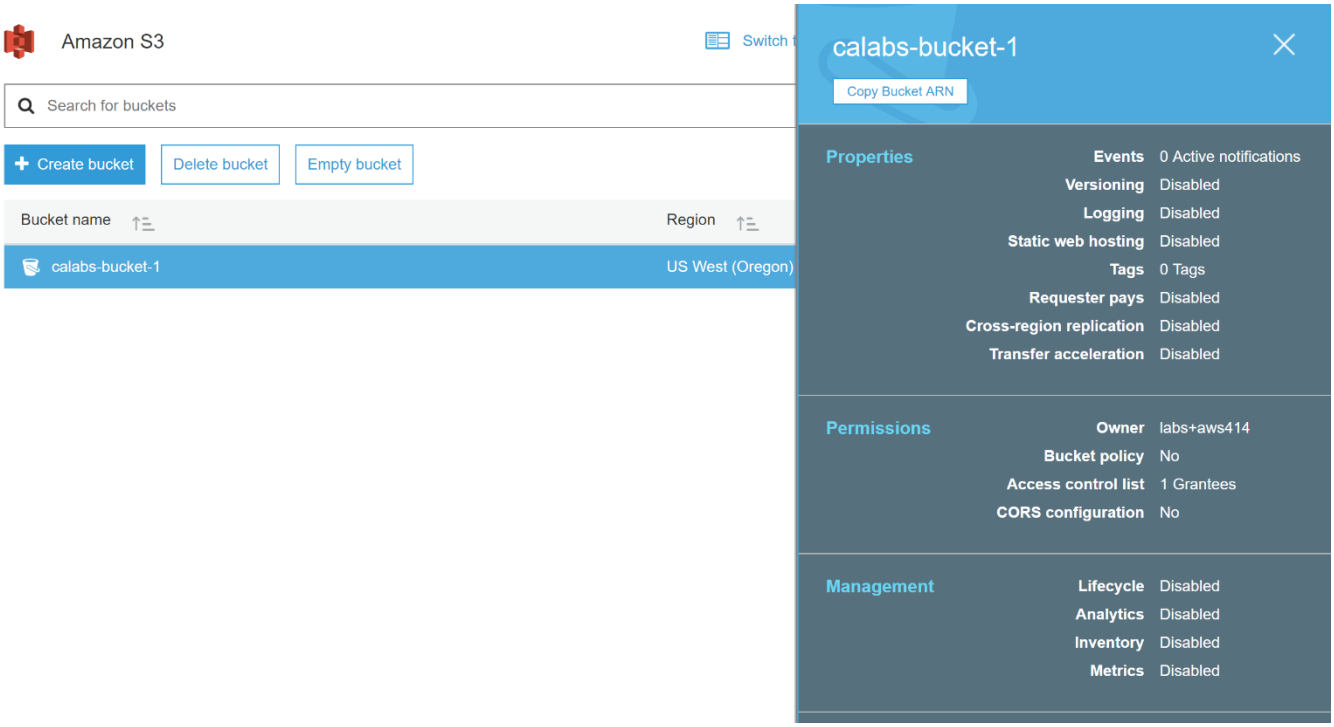

Create bucket
1 Name and region
2 Set properties
3 Set permissions
4 Review
Name and region
Bucket name
calabs-bucket
Region
US West (Oregon)
Copy settings from an existing bucket
You have no buckets
0 Buckets
Create
Cancel
Next

If you receive a "**The requested bucket name is not available**" error message, click **Previous** a few times in order to return to the original screen of the dialog, and append a unique number to the bucket name in order to guarantee its uniqueness:



For example, change "calabs-bucket" shown above to "calabs-bucket-1" (or other unique number) and click **Create** again. The console will display your empty bucket in the buckets list.

4. Select anywhere in the row that shows your bucket (but not on the actual name itself) in order to see additional information about your empty S3 bucket:



Step 2 Implement a Lambda Function to process S3 events

Now that you have a bucket ready to use, you can create a new Lambda Function and implement the processing logic.

The goal is to create a compressed version of every image uploaded into the S3 bucket.

You will save the original images with a prefix */images/* and the compressed objects with a prefix */zip/*. You can think of these prefixes as *S3 folders* within your bucket, which will allow us to avoid conflicts and infinite recursions. Indeed, if you listen to every creation event to generate new compressed files, you may end up with infinite loops. If new images are always uploaded to one folder, compressed and saved in another folder, then the looping error is avoided. In this simple case, prefixes are the best way to keep the Lambda Function simple and avoid problems.

First you will return to the main AWS Console page and select **AWS Lambda** (below Compute). Read the introductory text and then click **Get Started Now** when ready to proceed.

Set up the **Basic Information** page as follows:

- **Name**: *ZipS3Images*
- **Role**: Choose an *existing role*
- **Role Name**: *lambda_s3_write*

Fortunately, you can implement all the compression logic by using only Python built-in libraries, therefore you can simply edit the Lambda code in the browser.

Click **Create Function**.

Fill out the following sections:

Function Code:

- **Runtime:** *Python 2.7*
- **Handler**: *lambda_function.lambda_handler* (the convention is <FunctionName>.<HandlerName>)

Copy the Python code snippet below into the in-line code window (be sure to remove the 3 lines of default code in the window and just use the code below):

```
import os, zipfile, StringIO
import boto3

s3 = boto3.client('s3')

def lambda_handler(event, context):

    # read bucket and key from event data
    record = event['Records'][0]['s3']
    bucket = record['bucket']['name']
    key = record['object']['key']

    # generate new key name
    new_key = "zip/%s.zip" % os.path.basename(key)

    # read the source obj content
    body = s3.get_object(Bucket=bucket, Key=key)['Body'].read()

    # create new obj with compressed data
    s3.put_object(
        Body=compress(body, key),
        Key=new_key,
        Bucket=bucket,
    )
```

```
    return "OK"

def compress(body, key):
    data = StringIO.StringIO()
    with zipfile.ZipFile(data, 'w', zipfile.ZIP_DEFLATED) as f:
        f.writestr(os.path.basename(key), body)
    data.seek(0)
    return data.read()
```

Note: The code does not need any changes or substitutions. It will work as it is.

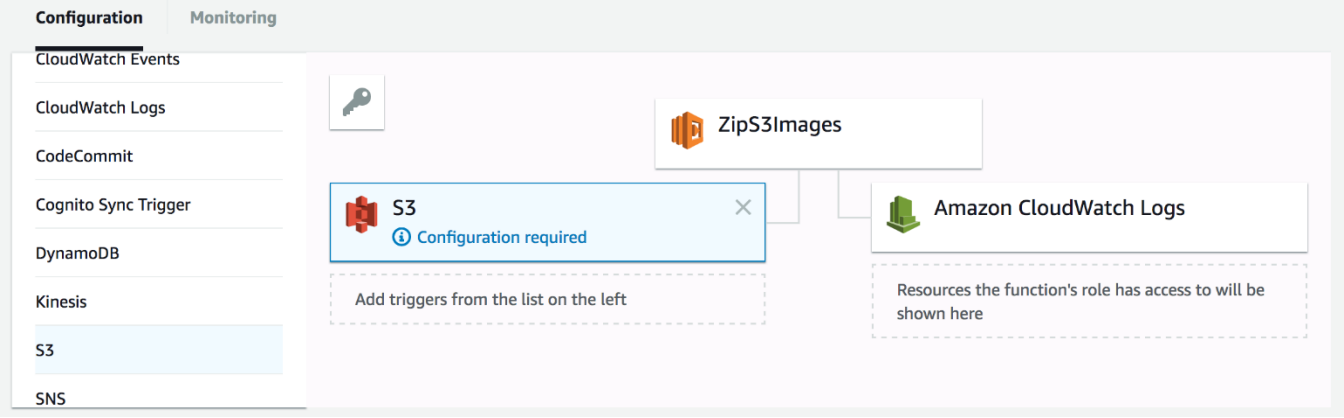
The *lambda_handler* function will take care of extracting the S3 Object information from the given event data. Then, it will read the object's body, compress it (in memory) and upload a new S3 Object with the same name into the */zip/* folder. The second function is just a simple Python utility to compress an input string with the ZIP algorithm.

Basic Settings

- **Timeout**: Set it to *0 min 10 sec*

Notice that the Lambda Function will require *read and write access* to the S3 bucket. Therefore, make sure you choose an existing role and select the **lambda_s3_write** execution role before confirming the Lambda Function creation.

Go to the **Triggers** box and select **S3** from the triggers list:

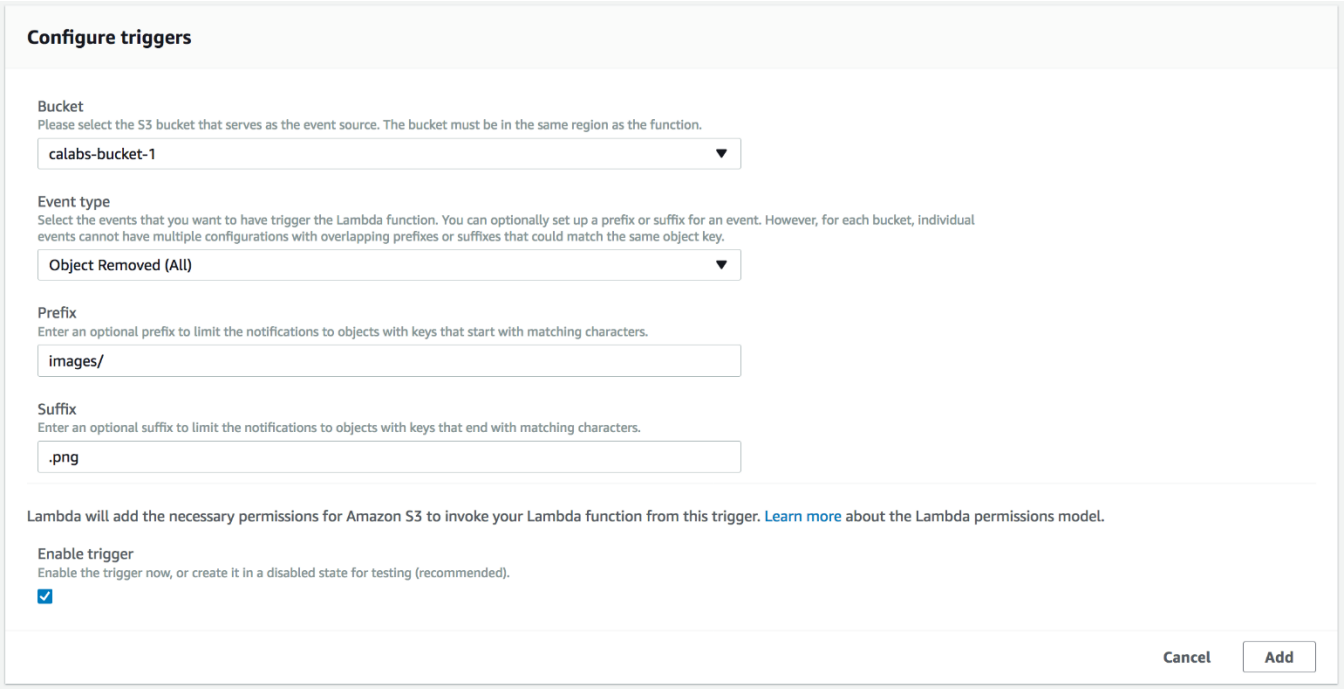


As mentioned previously, you will use a **prefix** to avoid infinite event recursion and a **suffix** to filter only png images. The Lambda Function will be invoked every time an object is created or updated, so that you can generate its compressed version. Next you will **Configure triggers**:

Configure the S3 trigger parameters as follows:

- **Bucket**: the unique bucket name you used
- **Event Type**: Object Created (All)
- **Prefix**: *images/* (Don't' forget to include a trailing slash (/))
- **Suffix**: *png*
- **Enable trigger:** Check the checkbox

Important! Don't forget to check the **Enable trigger** box before clicking **add** and then **save** in the right upper corner of the console.



In the next step, you will test the trigger and verify that zipped files are created whenever a png file is uploaded in the */images/* folder.

Step 3 Test the S3 Trigger

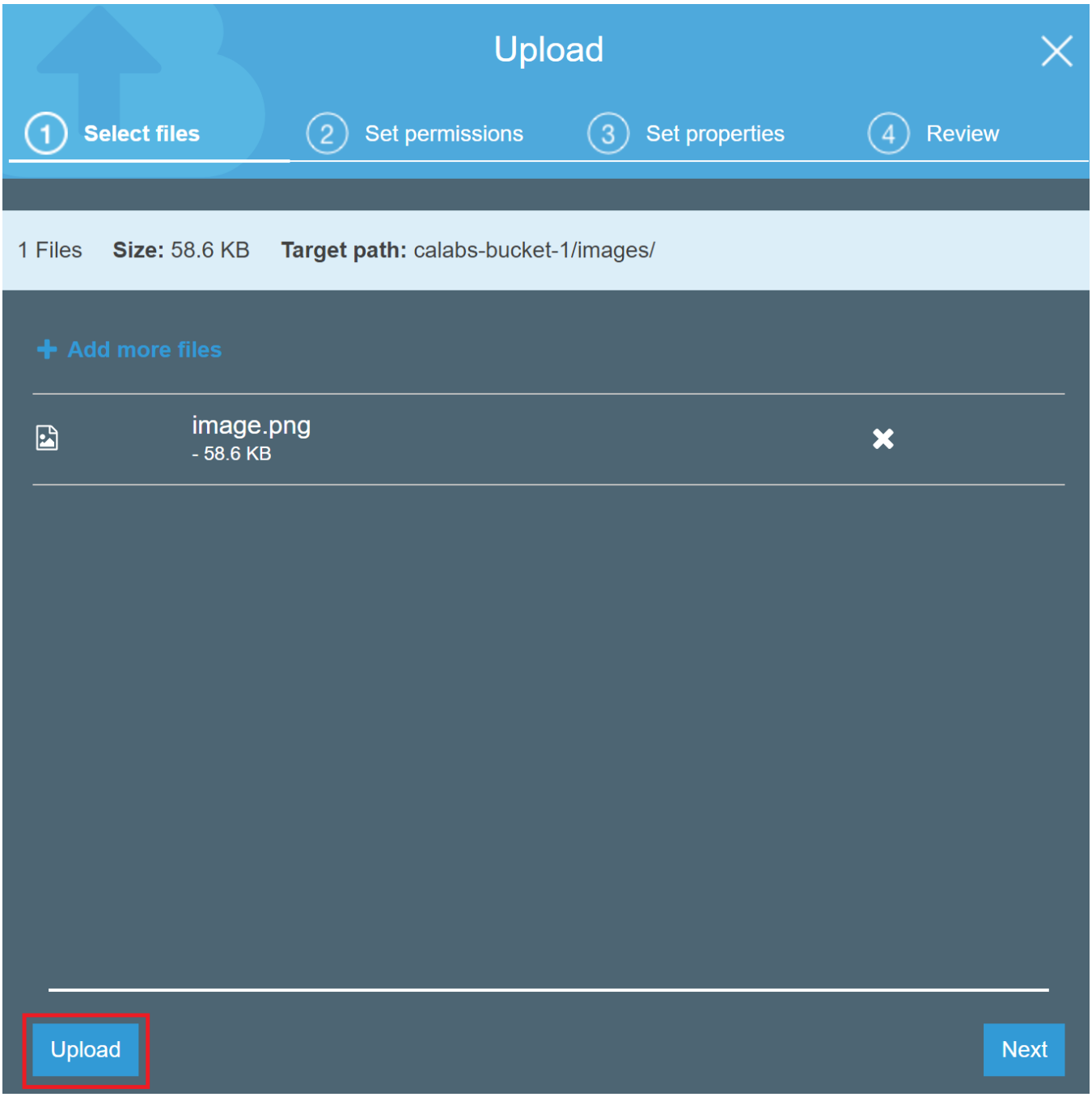
You will now quickly test that your trigger is working properly.

Switch to S3 again, click on your bucket name and create a newfolder named *images* by clicking the **+ Create Folder** button and entering *images* in the **New folder** field.

Select the *images* folder in order to navigate into it and then click **Upload**.

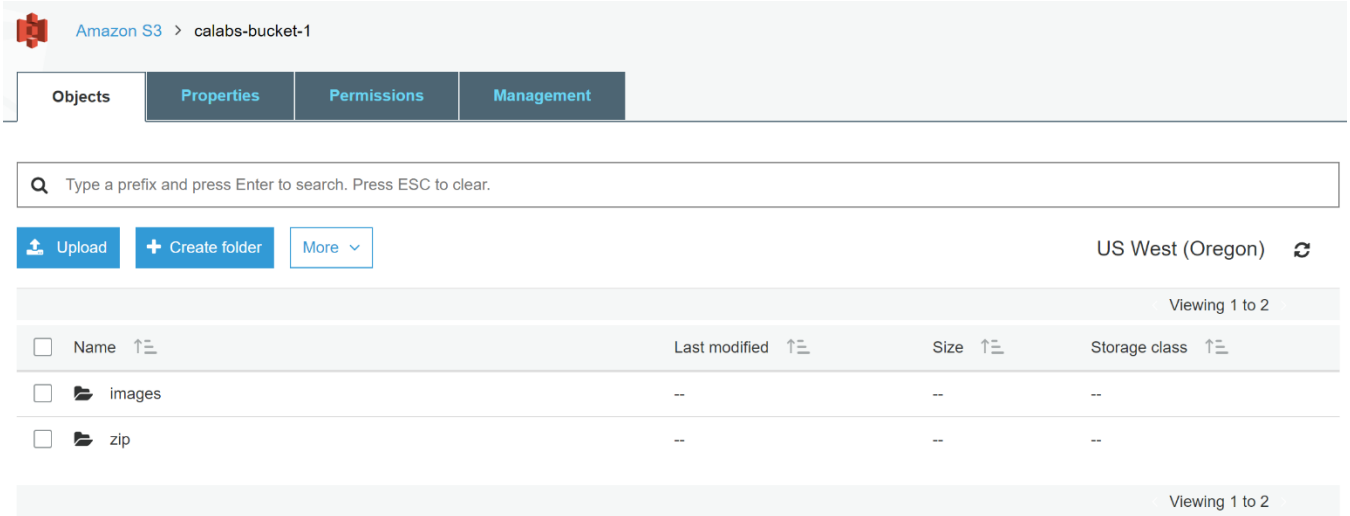
Important! *Upload PNG files that have no spaces in them.* The code only recognizes file names with the following convention: <*filename*>.png A flle named "My file.png" will not throw an error, but Lambda will never pick up the file, compress it and drop it into the /zip/ folder.

Here you will be asked to upload some files. You can drag and drop or click **Add Files** and select one or more png files (with no spaces in the file names). Once you are ready, you can start the upload by clicking **Upload**:



The AWS Lambda invocation will be completely transparent and you will not notice anything until you examine S3 further.

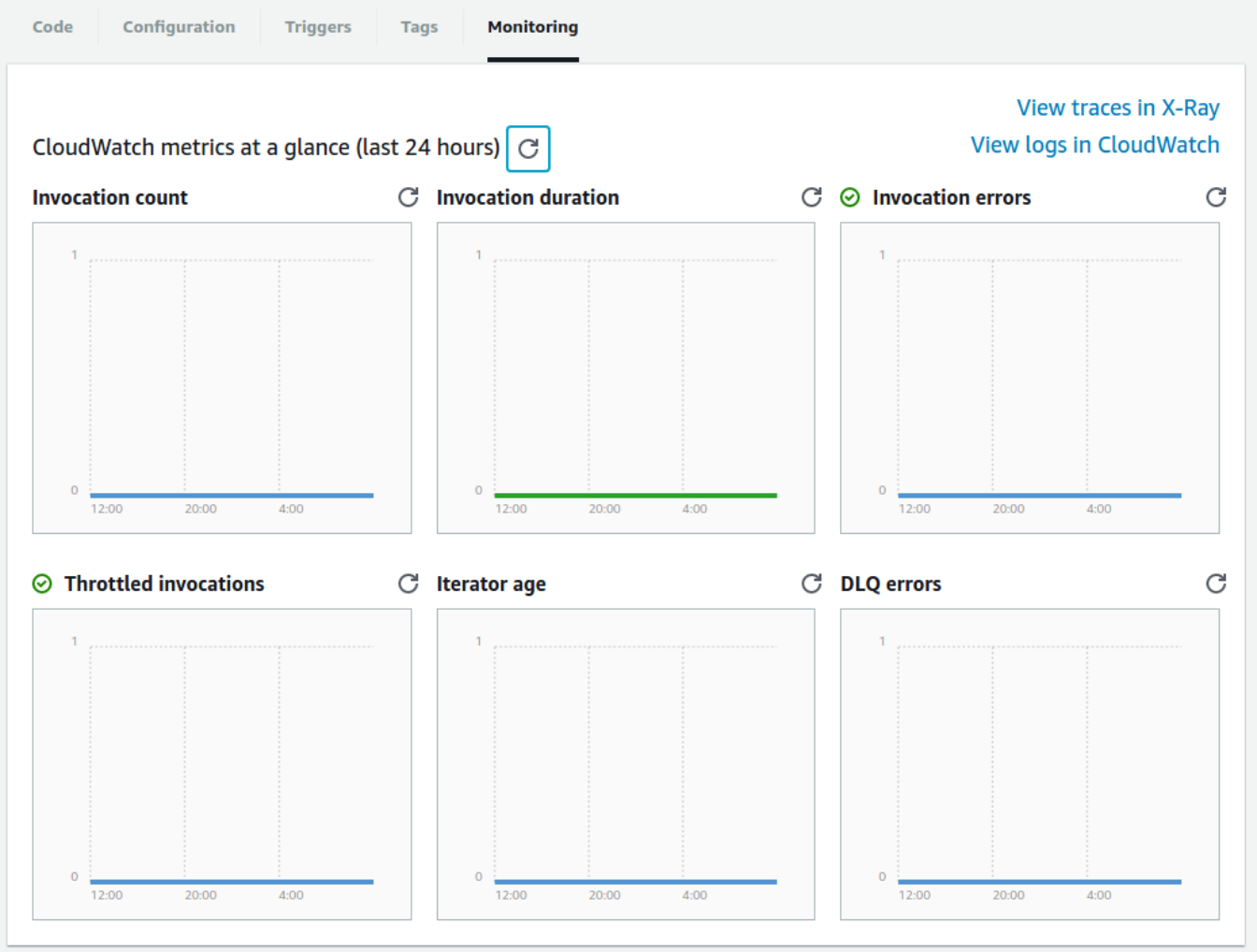
If everything is correctly configured, you will be able to inspect the bucket root and find a new *zip* folder with a new file inside (*image.png.zip*):



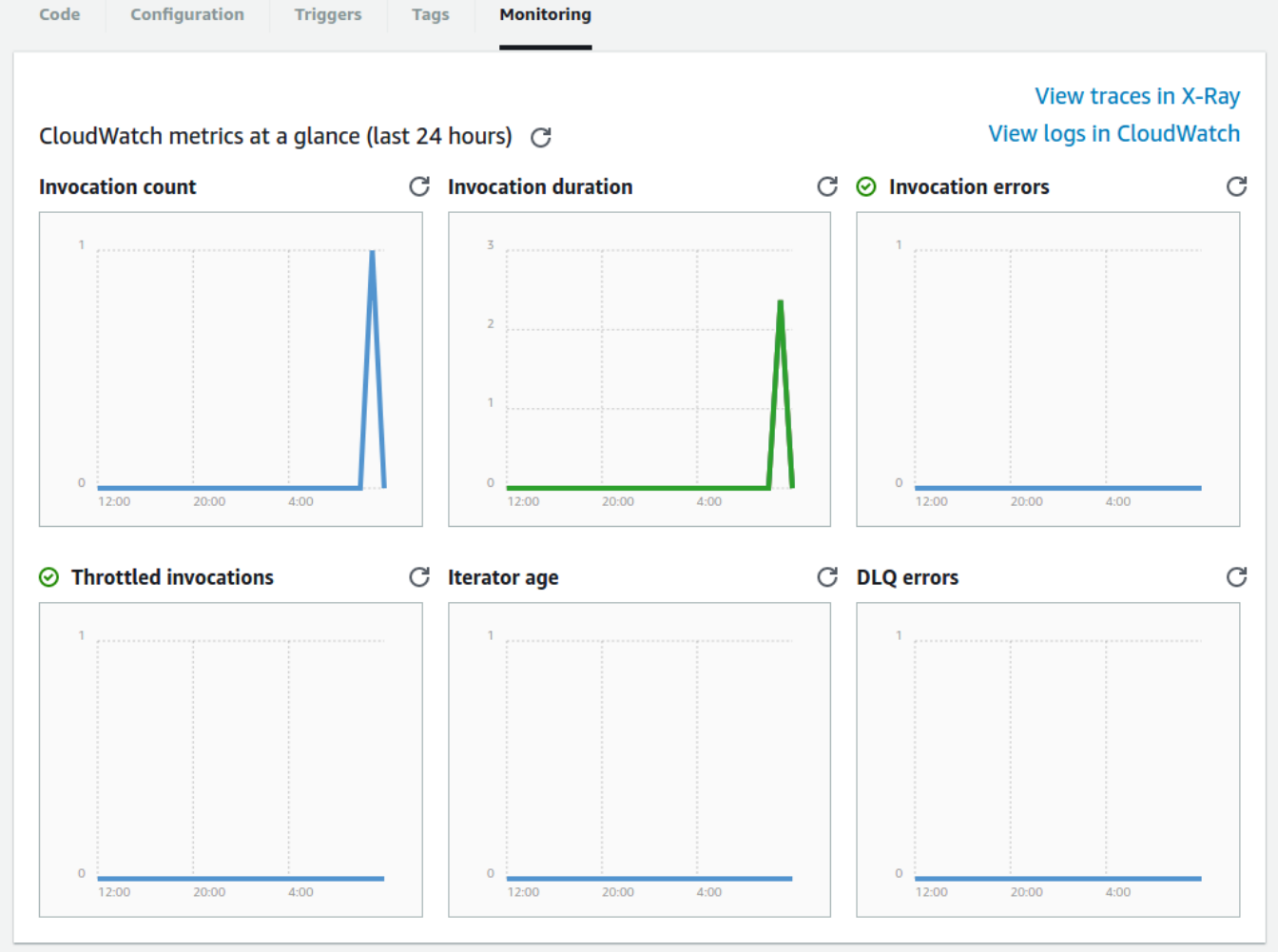
You can select the file and download it (**Actions > Download**) in order to verify its content, which should be exactly the same as the original file when unzipped.

In case you can't see the new folder, just wait a few seconds and click on the refresh button (top-right). If nothing appears, you may want to verify your Lambda Function configuration and ensure you didn't miss anything in the previous step. For example, make sure your PNG files do not include spaces in their name, and that your trigger is enabled. The */zip/* folder will get created for you, so you should not have to worry about creating it beforehand in S3.

Consider looking at the **Monitoring** tab as well. Before Lambda runs, it will be flat lined:



With each run, you should of course see the **Invocation count** increment. The **Invocation duration** will have a non-zero length to it as well:



Hopefully, the Invocation errors remain at zero!