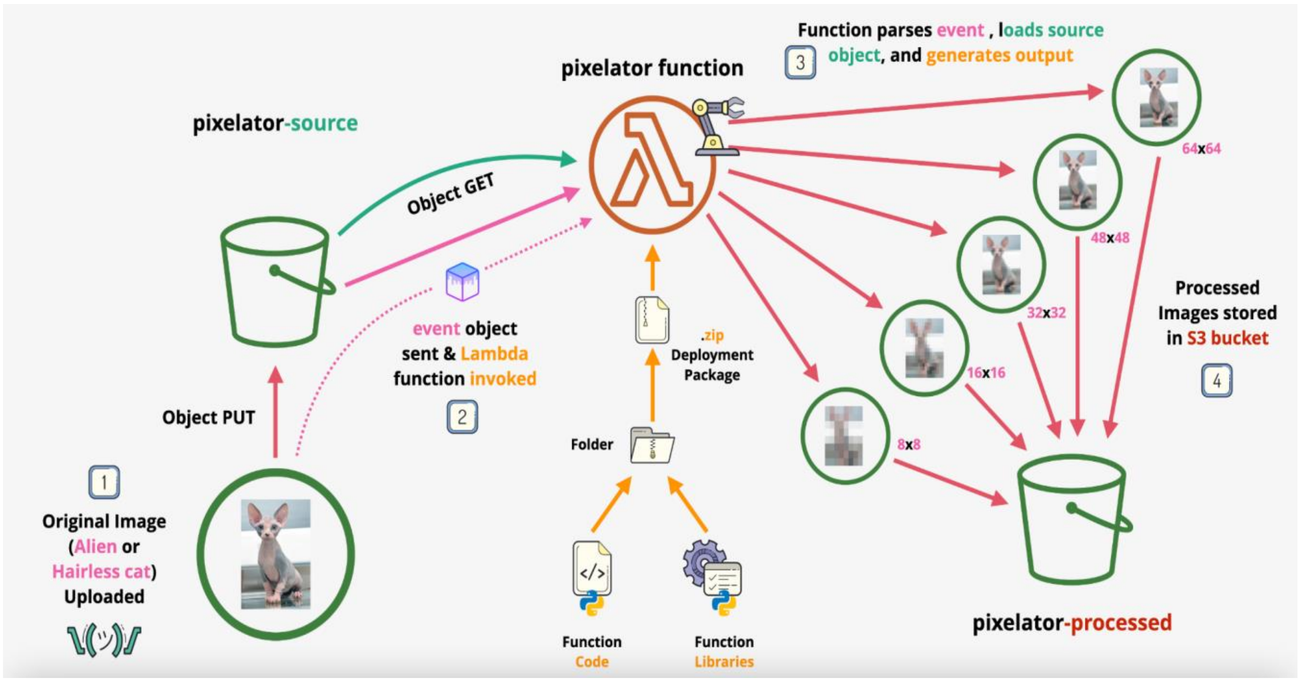
Event Driven Architecture:

An event-driven architecture uses events to trigger and communicate between decoupled services and is common in modern applications built with microservices. An event is a change in state, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).

Event-driven architectures have three key components: event producers, event routers, and event consumers. A producer publishes an event to the router, which filters and pushes the events to consumers. Producer services and consumer services are decoupled, which allows them to be scaled, updated, and deployed independently.

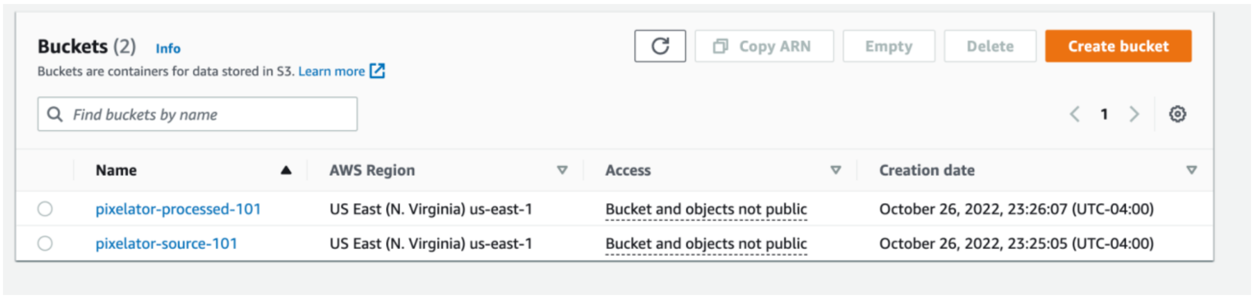
In this demo we are going to create a simple event-driven image processing pipeline. The pipeline uses two S3 buckets, a source bucket and a processed bucket. When images are added to the source bucket a lambda function is triggered based on the PUT. When invoked the lambda function receives the event and extracts the bucket and object information. Once those details are known, the lambda function, using the PIL module pixelates the image with 5 different variations (8x8, 16x16, 32x32, 48x48 and 64x64) and uploads them to the processed bucket.

Demo Architecture


Function parses event, loads source object, and generates output
3
pixelator function
pixelator-source
Object GET
event object sent & Lambda function invoked
2
Object PUT
1
Original Image (Alien or Hairless cat) Uploaded
.zip Deployment Package
Folder
Function Code
Function Libraries
64x64
48x48
32x32
16x16
8x8
Processed Images stored in S3 bucket
4
pixelator-processed

Step1:

Created two S3 buckets as shown in the image below.



Buckets (2) Info

Buckets are containers for data stored in S3. Learn more

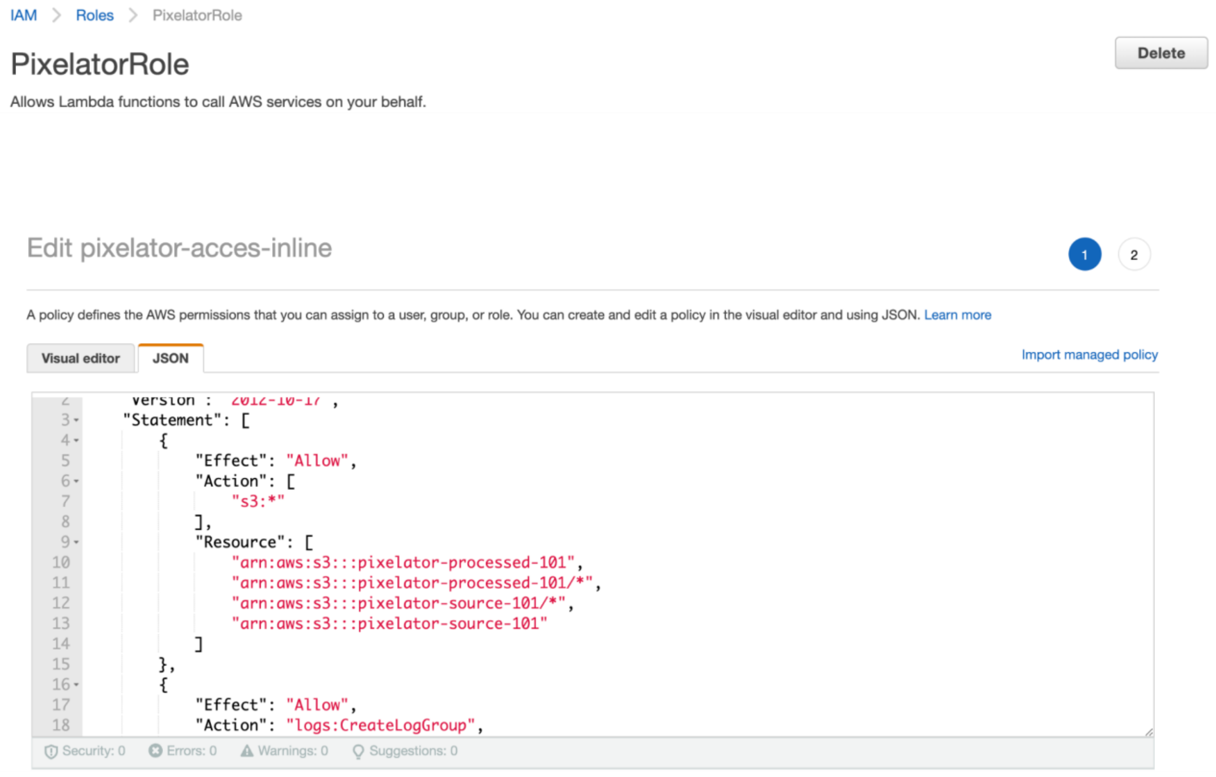
	Name	AWS Region	Access	Creation date
○	pixelator-processed-101	US East (N. Virginia) us-east-1	Bucket and objects not public	October 26, 2022, 23:26:07 (UTC-04:00)
○	pixelator-source-101	US East (N. Virginia) us-east-1	Bucket and objects not public	October 26, 2022, 23:25:05 (UTC-04:00)

Image1: Two buckets created

Two different buckets named pixelator-processed-101 and pixelator-source-101 are created. As the name suggests, a pixelator-processed-101 bucket is used to store the processed image coming in after lambda execution. And, pixelator-source-101 is used to put the image which needs to be processed.

Step2:

Created a Lambda Role with inline-policy that allows Lambda to interact with two mentioned S3 buckets. As shown in the image below, a Lambda role and inline policy is attached to it.



IAM > Roles > PixelatorRole

PixelatorRole

Allows Lambda functions to call AWS services on your behalf.

Edit pixelator-acces-inline

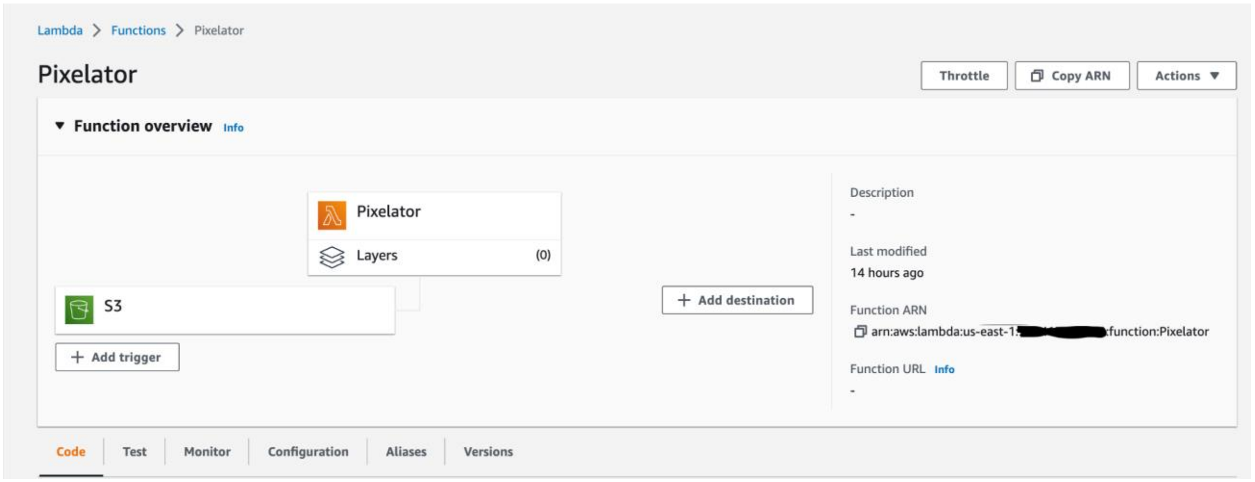
A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. Learn more

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "arn:aws:s3:::pixelator-processed-101",
                "arn:aws:s3:::pixelator-processed-101/*",
                "arn:aws:s3:::pixelator-source-101/*",
                "arn:aws:s3:::pixelator-source-101"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
```

Image2: Iam-Role with Inline policy

Step3:

Created Lambda function with the proper assigned role as we created it earlier. And, trigger for the Lambda function is created as s3 bucket and the name of the bucket is pixelator-source-101 is selected. Also, the code for processing images into different pixelets is placed in the Lambda function.
So, when an object in s3 bucket named pixelator-source-101 is kept, it will generate a trigger for the lambda function named Pixelator and then the Lambda function will process the image as per the code written. Once the code is executed on the incoming image, it will send the final results into the destination bucket which we mentioned there as pixelator-processed-101.



Description
-

Last modified
14 hours ago

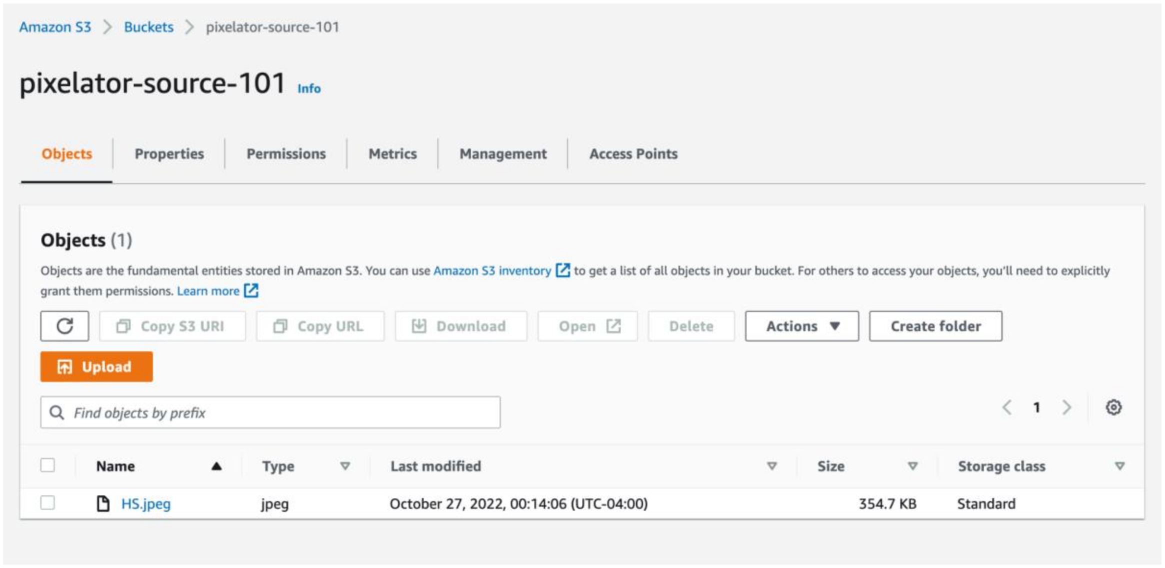
Function ARN
arn:aws:lambda:us-east-1[redacted]function:Pixelator

Function URL Info
-

Image3: Lambda Function created

Step4:

Test and monitor: In the end we check if our integration worked as expected or not. For testing purposes, I uploaded a picture named HS.jpeg. Once it is uploaded to the bucket named pixelator-source-101 bucket, it fires the trigger and our Lambda function gets called. The computation mentioned in the Lambda function gets performed and it then converts the image to a different size and uploads it into another bucket named pixelator-processed-101.



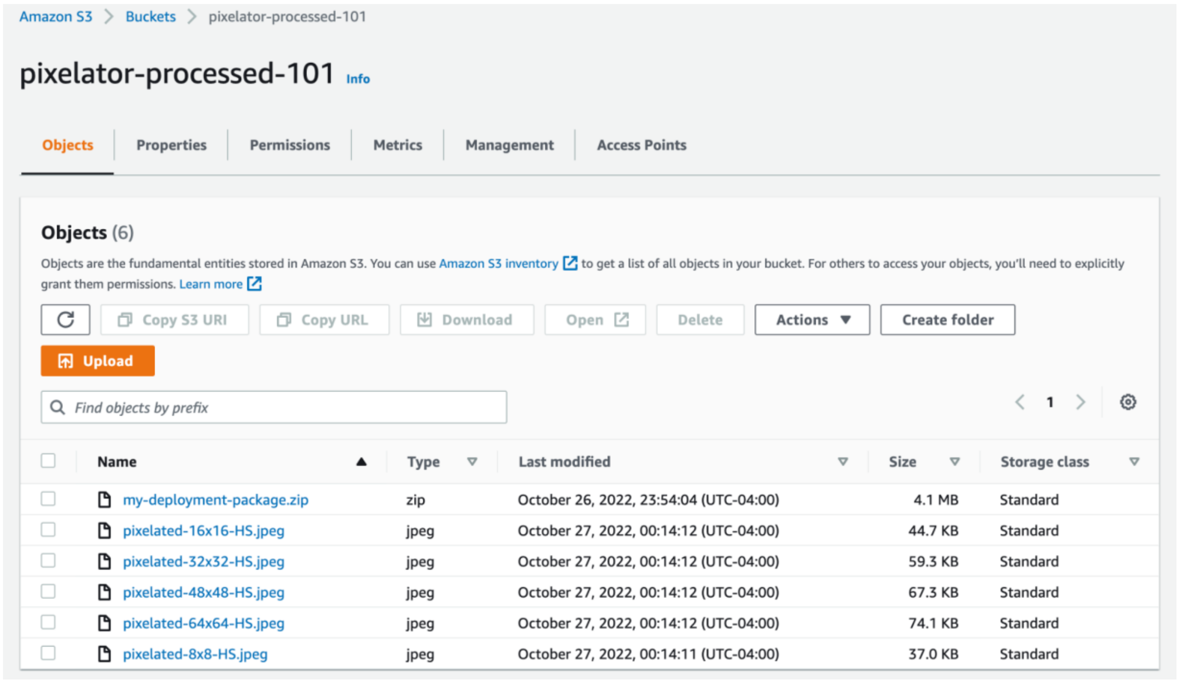
pixelator-source-101

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
HS.jpeg	jpeg	October 27, 2022, 00:14:06 (UTC-04:00)	354.7 KB	Standard

Image4: Source bucket



pixelator-processed-101

Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
my-deployment-package.zip	zip	October 26, 2022, 23:54:04 (UTC-04:00)	4.1 MB	Standard
pixelated-16x16-HS.jpeg	jpeg	October 27, 2022, 00:14:12 (UTC-04:00)	44.7 KB	Standard
pixelated-32x32-HS.jpeg	jpeg	October 27, 2022, 00:14:12 (UTC-04:00)	59.3 KB	Standard
pixelated-48x48-HS.jpeg	jpeg	October 27, 2022, 00:14:12 (UTC-04:00)	67.3 KB	Standard
pixelated-64x64-HS.jpeg	jpeg	October 27, 2022, 00:14:12 (UTC-04:00)	74.1 KB	Standard
pixelated-8x8-HS.jpeg	jpeg	October 27, 2022, 00:14:11 (UTC-04:00)	37.0 KB	Standard

Image5: Processed bucket

Thank You
Vaishal Shah