Managed Cloud Services

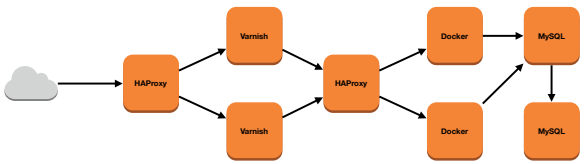
When you don't want to run it yourself

Managed Cloud Services

Virtual Servers vs Cloud Services

- All the pieces of internet applications began as discrete software run on a server you managed
- Everyone had to be at least an intermediate level sysadmin
- Managed Cloud Services aim to take away the "undifferentiated heavy lifting" from your application stack

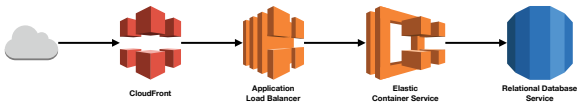
VM Centric Architecture



Managed Cloud Services

	VM / EC2	AWS Service
Database	MySQL	RDS MySQL
Load Balancer	HAProxy	Elastic Load Balancer Application Load Balancer
Docker	Docker	Elastic Container Service
Cacheing	Varnish	CloudFront

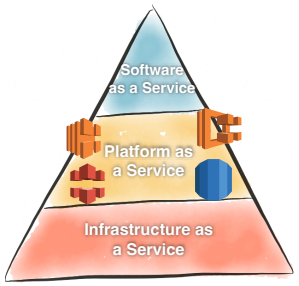
Cloud Centric Architecture



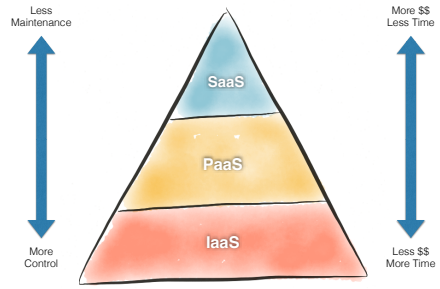
Managed Cloud Services

Virtual Servers vs Cloud Services

- All these AWS services are highly available, fault tolerant, and can be automatically deployed and backed up
- Only the RDS instance needs to be updated, and ~80% of that is automatic



7



8

AWS S3

Simple Storage Service

9

AWS S3

Cloud Object Storage

- Amazon S3 is an object storage service that stores data as objects within buckets.
- An object is a file and any metadata that describes the file.
- A bucket is a container for objects.
- Not a File System
- Read/Write object data through AWS API

AWS S3

Cloud Object Storage

- Bucket names must be globally unique
- No size limits
- Objects can be public or private
- Public objects can have URLs for direct access
  - This makes S3 ideal for storing data on the internet you want other people to access.

AWS S3

S3 Public Website Bucket

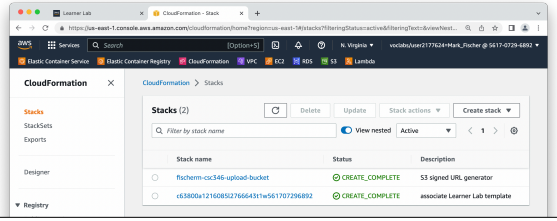
- There are enough little things that need to be configured on an S3 bucket to allow for public web access that I built a CloudFormation template to codify it.
- AWS has a full tutorial for this:

```
https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html
```

AWS S3

S3 Public Website Bucket

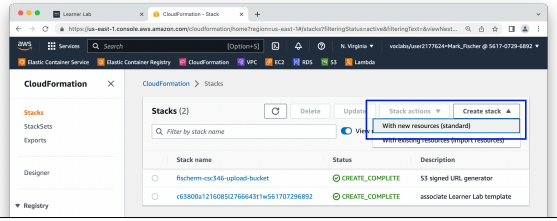
- To deploy the template, go to the CloudFormation console in the web UI.

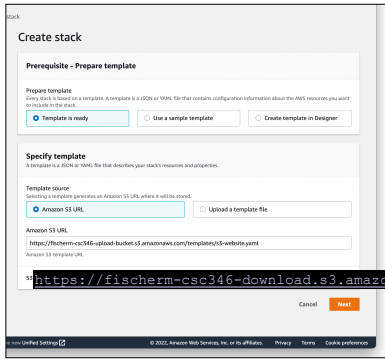


AWS S3

S3 Public Website Bucket

- Create a new stack with new resources



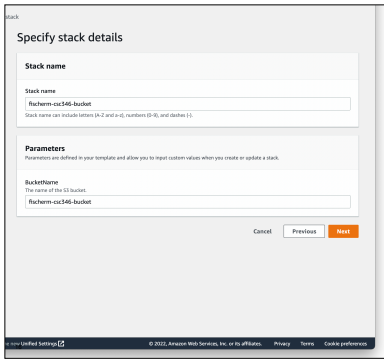


AWS S3

S3 Public Website Bucket

- You can use my template directly from my class bucket.

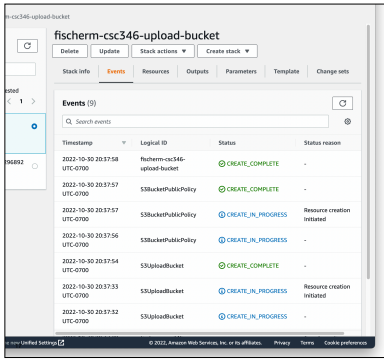
```
https://fischerm-csc346-download.s3.amazonaws.com/s3_template.yaml
```



AWS S3

S3 Public Website Bucket

- You need to specify a Stack name
- There's one parameter for this template, the bucket name
- I often have the stack name and bucket name be the same. Makes things simple
- Create a unique bucket name!



AWS S3

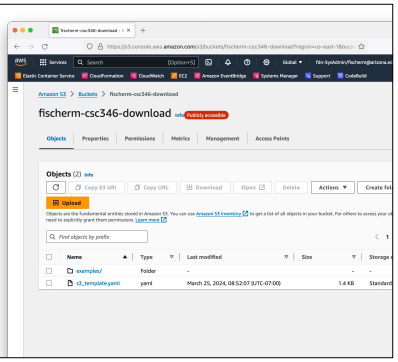
S3 Public Website Bucket

- Click through to deploy the stack
- Once the stack reaches CREATE_COMPLETE your S3 bucket should be created and configured correctly to host files able to be accessed publicly.
- We will use this in an upcoming homework to store images for our chat app.

AWS S3

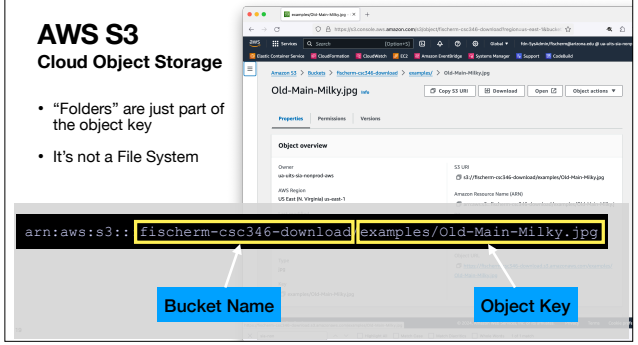
Cloud Object Storage

- Clicking on a bucket shows its contents
- Can create "folders" and upload objects directly in the web UI



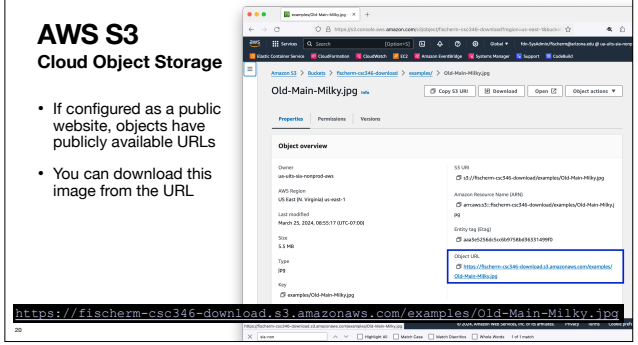
AWS S3
Cloud Object Storage

- "Folders" are just part of the object key
- It's not a File System



AWS S3
Cloud Object Storage

- If configured as a public website, objects have publicly available URLs
- You can download this image from the URL



AWS S3
Cloud Object Storage

- S3 underpins much of AWS
- Docker images in ECR are stored in S3 under the hood
- All CloudFormation templates you upload are stored in an S3 bucket
- All EC2 AMI images are stored in S3
- It is a really important service!

AWS S3

Too many features to go over in class

- Storage tiers - save money if you accept more risk
- Lifecycle Policies - Delete stuff after a while, or transition it to archive storage
- Integrates with many other Services - Event Based Triggers
- Cross-account access - Host files that others can use
- Requestor-pays - Host files that others have to pay to download (they don't pay you, they pay the AWS S3 network costs)
- Yes, you have to pay to read data out of S3, that's where they getcha!

AWS Lambda

Function as a Service?

AWS Lambda

Managed Code Execution

- Up to this point, if we had code we needed to execute, it had to run on a machine we managed.
  - Laptop
  - EC2
- AWS Lambda introduces another model

AWS Lambda

Managed Code Execution

"Lambda is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging."

`https://docs.aws.amazon.com/lambda/latest/dg/welcome.html`

AWS Lambda

Advantages

- Serverless - No infrastructure to manage
- Event-Driven - Nothing is "always running" (this can be good and bad)
- Pricing based only on what you use
- Scales automatically (can have limits placed)
- Can be massively parallelized
- Lets you focus on just your core application logic

AWS Lambda

Disadvantages

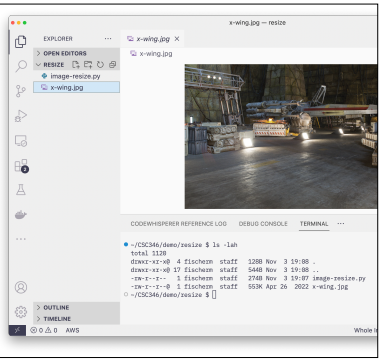
- Not for long-running processes. A given Lambda invocation cannot last longer than 15 minutes.
- Requires a different mental model for how you build an application.
  - Micro-services vs monolithic services.
- Vendor lock-in. Can't really take your AWS Lambda functions to Google App Engine.
- Memory and CPU limits are not as high as dedicated EC2 instances.
- Access to persistent file systems is not simple.

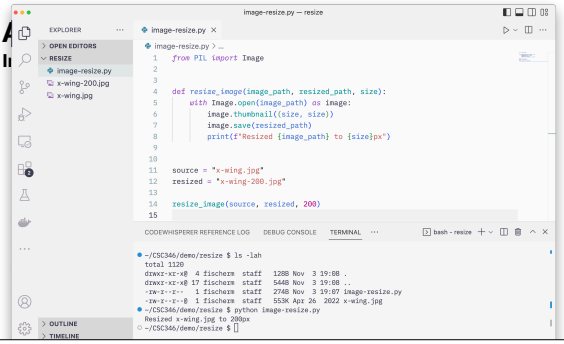
AWS Lambda
Image Resizing

- Let's add images to our app.
- Images are uploaded of all sorts of various sizes.
- In the posts list, we want the images to all be a uniform size.
- We want to normalize any uploaded image to be a set of standard sizes, a square thumbnail and a larger view, but still possibly smaller than the original image.

AWS Lambda
Image Resizing in Python

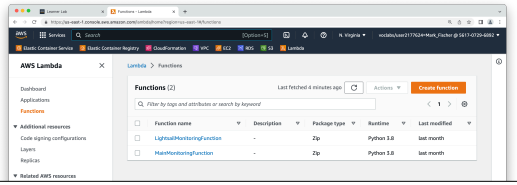
- How do we resize an image in Python?
- Use the Pillow / PIL module





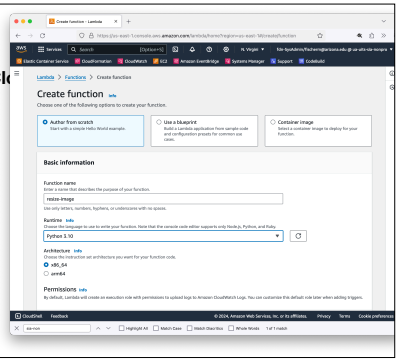
AWS Lambda
Image Resizing in the Cloud

- That's all fine for a laptop, how do we do this in the cloud?
- AWS Lambda Console - Search for Lambda



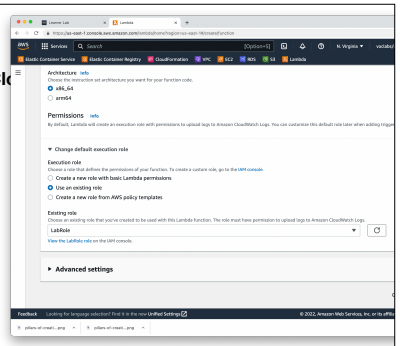
AWS Lambda
Image Resizing in the Cl

- Create a new function
- Give it a name
- Use python 3.10 for the runtime
- x86_64



AWS Lambda
Image Resizing in the Cl

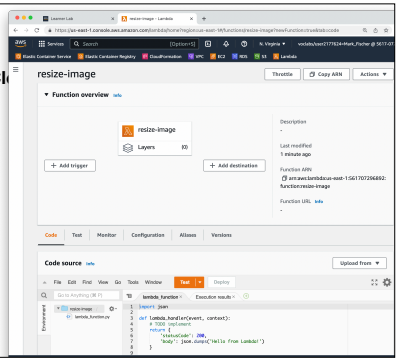
- Change the default execution role
- We can't make new IAM roles in the Academy account
- Use the existing "LabRole"



AWS Lambda

Image Resizing in the Cl

- Default “Hello World” function



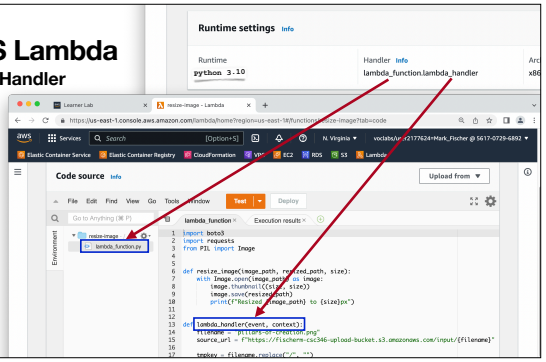
AWS Lambda

Event Handler

- We mentioned that Lambda is event driven
- Your code runs inside of the Lambda Runtime
- The Lambda Runtime handles receipt of events, then calls your code and passes the event to it
- The entry point to your code is your event handler function

AWS Lambda

Event Handler



AWS Lambda
Event Triggers

- So what is in an event?
- It's largely dependent on what is triggering your Lambda Function
- So what can trigger Lambda?
  - In short, a lot of things!
- Most basic trigger is direct invocation. Either in the web console, or with the API

```
aws lambda invoke --function-name resize-image --payload '{ "file": "x-wing.jpg" }'
```


AWS Lambda
Event Triggers

- Lambda integrates with more than 140 AWS services via direct integration and the Amazon EventBridge event bus.
- Commonly used Lambda event sources:
  - API Gateway
  - SNS
  - SQS
  - S3
  - CloudWatch Logs
  - CloudWatch Events
  - DynamoDB
  - EventBridge
  - Kinesis Data Streams
  - Step Functions

AWS Lambda
Event Triggers

- Each event source will send different bits of data in the incoming `event` object.
- Here is a sample event coming from API Gateway
- Data relevant to an incoming HTTP REST call


```
{
    "resource": "/",
    "path": "/",
    "httpMethod": "GET",
    "requestContext": {
        "resourcePath": "/",
        "httpMethod": "GET",
        "path": "/Prod/",
        ...
    },
    "headers": {
        "accept": "text/html,application/signed-exc
        "accept-encoding": "gzip, deflate, br",
        "Host": "70ixmpl4fl.execute-api.us-east-2.a
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0
        ...
    },
    "queryStringParameters": null,
    "multiValueQueryStringParameters": null,
    "pathParameters": null,
    "stageVariables": null,
    "body": null,
    "isBase64Encoded": false
}
```

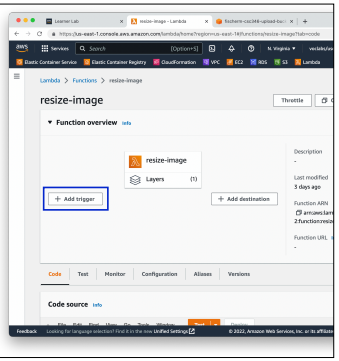
AWS Lambda
Event Triggers

- Here's an example of an S3 `ObjectCreated:Put` event
- Information about which bucket the object was created in as well as the object itself
- Note that the `Records` key in the top level dictionary is an array. This event may container multiple objects


```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "2022-11-06T20:17:18.352Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AROAYFSC5FB6KLKFWGIOO:user2177624=Ma
      },
      "requestParameters": {"sourceIPAddress": "67.1.196.37"},
      "responseElements": {
        "x-amz-request-id": "VV31VSKAKPTP7R4C",
        "x-amz-id-2": "v+A+vGX30SWO8cb8JhbAj7wRPmtDLn1dgYtZQOf9Z
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "e2733ff1-399f-4645-8778-7e4fef7a7c3a
        "bucket": {
          "name": "fischerm-csc346-upload-bucket",
          "ownerIdentity": {"principalId": "A3NRT1KH8KAG57"},
          "arn": "arn:aws:s3:::fischerm-csc346-upload-bucket",
        },
        "object": {
          "key": "input/x-wing.jpg",
          "size": 566695,
          "eTag": "09a9b11f91823dd69fefc3ecbd9f7e9c",
          "sequencer": "006368164E4491ED05",
        },
      },
    }
  ]
}
```

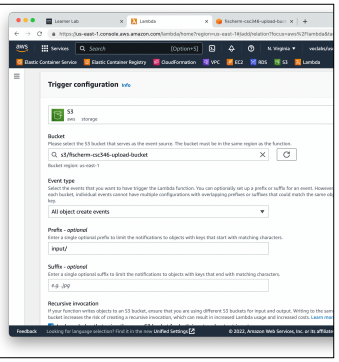

AWS Lambda
Event Triggers

- In the Lambda console, click "Add trigger"



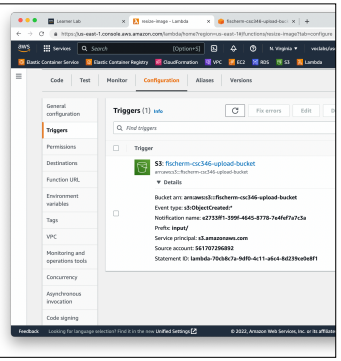
AWS Lambda
Event Triggers

- Choose S3 as the event source
- Select the S3 bucket you want
- We'll trigger on all the "CreateObject" events
- I only want to trigger on objects with keys beginning with "input/"
- Be careful about recursive triggering!!



AWS Lambda
Event Triggers

- Once saved, you can see the trigger configuration in the “Configuration” tab of your function
- Now every time a new object is created in the input folder of that bucket, our Lambda function will run!



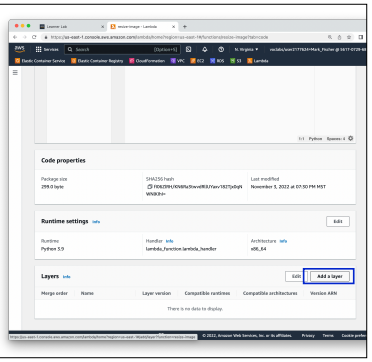
AWS Lambda
Layers

- How do we import all the various python modules, such as the Pillow/PIL module?
- Lambda supports the idea of shared layers.
- I’ve created a layer which has all the dependencies built in.
- Layers aren’t too hard to create, but we don’t have enough time to go into that in class unfortunately.
- Only available in the same region, so use `us-east-1`

```
arn:aws:lambda:us-east-1:269800669561:layer:fischerm-csc346-imagelayer:2
```

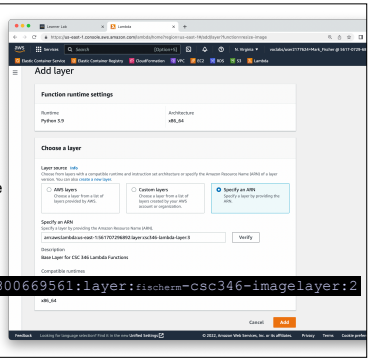

AWS Lambda
Layers

- Scroll down
- Click the “Add a layer” button



AWS Lambda
Layers

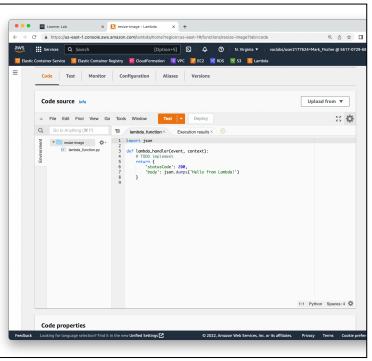
- Specify an ARN
- Use my layer ARN
- Click the Verify button to make sure things are working
- Click the Add button




```
arn:aws:lambda:us-east-1:269800669561:layer:fischerm-csc346-imagelayer:2
```

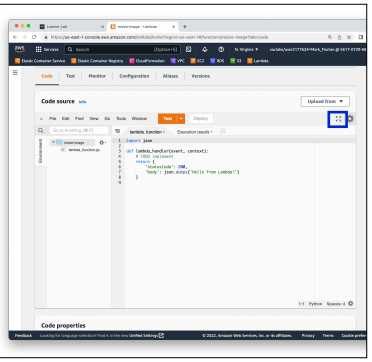

AWS Lambda
Layers

- We can edit the code directly in the browser to start.
- Works for simple functions.
- OK for testing.
- You'll want to have more Infrastructure as Code scaffolding around any real project.



AWS Lambda
Layers

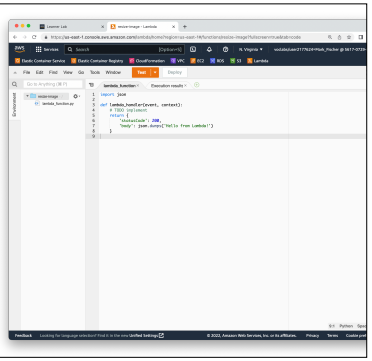
- Can make the code editor fill the browser window



AWS Lambda

Layers

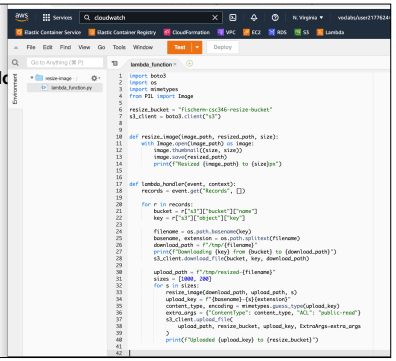
- Can make the code editor fill the browser window



AWS Lambda

Image Resizing in the Cl

- Where are our files?
- The Lambda runtime has access to some temporary local storage
- We need to get the file to resize from the event when a new object is added to the bucket

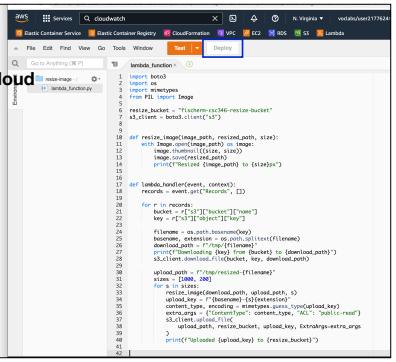


```
import boto3
import os
import mimetypes
from PIL import Image

resize_bucket = "fischerm-csc346-resize-bucket"
s3_client = boto3.client("s3")


def resize_image(image_path, resized_path, size):
    with Image.open(image_path) as image:
        image.thumbnail((size, size))
        image.save(resized_path)
        print(f"Resized {image_path} to {size}px")


def lambda_handler(event, context):
    records = event.get("Records", [])

    for r in records:
        bucket = r["s3"]["bucket"]["name"]
        key = r["s3"]["object"]["key"]

        filename = os.path.basename(key)
        basename, extension = os.path.splitext(filename)
        download_path = f"/tmp/{filename}"
        print(f"Downloading {key} from {bucket} to {download_path}")
        s3_client.download_file(bucket, key, download_path)

        upload_path = f"/tmp/resized-{filename}"
        sizes = [1000, 200]
        for s in sizes:
            resize_image(download_path, upload_path, s)
            upload_key = f"{basename}-{s}{extension}"
            content_type, encoding = mimetypes.guess_type(upload_key)
            extra_args = {"ContentType": content_type, "ACL": "public-read"}
            s3_client.upload_file(
                upload_path, resize_bucket, upload_key, ExtraArgs=extra_args
            )
            print(f"Uploaded {upload_key} to {resize_bucket}")
```


AWS Lambda

Image Resizing in the Cloud

- Function needs to be Deployed before testing.

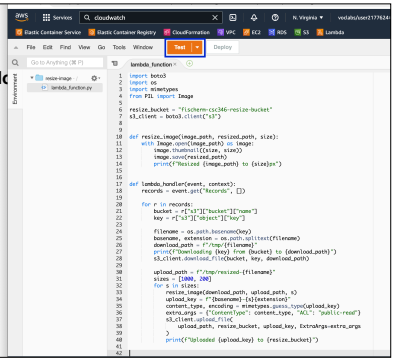


```
import boto3
import os
import mimetypes
from PIL import Image

resize_bucket = "fischerm-csc346-resize-bucket"
s3_client = boto3.client("s3")


def resize_image(image_path, resized_path, size):
    with Image.open(image_path) as image:
        image.thumbnail((size, size))
        image.save(resized_path)
        print(f"Resized {image_path} to {size}px")


def lambda_handler(event, context):
    records = event.get("Records", [])

    for r in records:
        bucket = r["s3"]["bucket"]["name"]
        key = r["s3"]["object"]["key"]

        filename = os.path.basename(key)
        basename, extension = os.path.splitext(filename)
        download_path = f"/tmp/{filename}"
        print(f"Downloading {key} from {bucket} to {download_path}")
        s3_client.download_file(bucket, key, download_path)

        upload_path = f"/tmp/resized-{filename}"
        sizes = [1000, 200]
        for s in sizes:
            resize_image(download_path, upload_path, s)
            upload_key = f"{basename}-{s}{extension}"
            content_type, encoding = mimetypes.guess_type(upload_key)
            extra_args = {"ContentType": content_type, "ACL": "public-read"}
            s3_client.upload_file(
                upload_path, resize_bucket, upload_key, ExtraArgs=extra_args
            )
            print(f"Uploaded {upload_key} to {resize_bucket}")
```

AWS Lambda

Image Resizing in the Cl

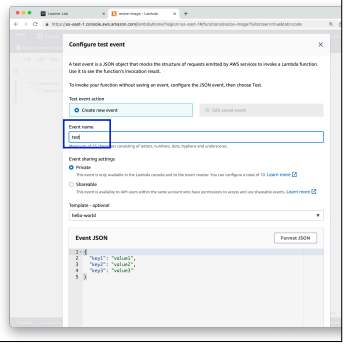
- Once deployed, we can Test



AWS Lambda

Image Resizing in the Cloud

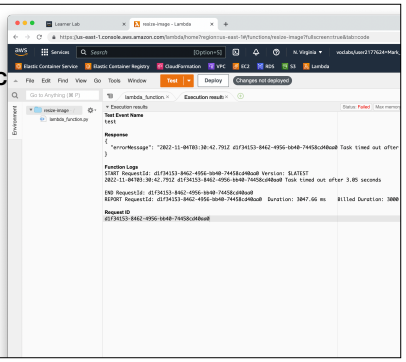
- The first time we hit Test, we're prompted to define a Test Event
- Lambda is Event Driven
- Our function currently doesn't use the event at all, so the default "hello-world" event is fine
- Give it an Event name
- Scroll down and Save



AWS Lambda

Image Resizing in the C

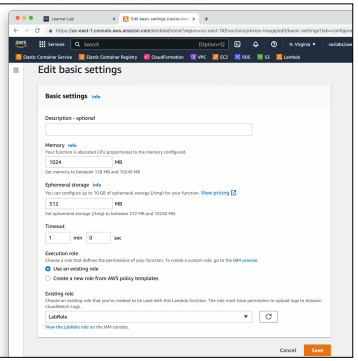
- Try testing again
- Error!
- Task timed out after 3 seconds?
- Lambda functions can last *up to 15 minutes*, but default to 3 seconds.



AWS Lambda

Image Resizing in the Cloud

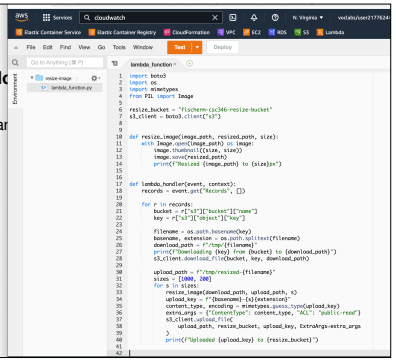
- Memory size is also tied to CPU allocation. Let's raise the memory limit ti 1024, that gives us more CPU and our function will run faster
- Change the Timeout to 1 minute.
- Save



AWS Lambda

Image Resizing in the Clo

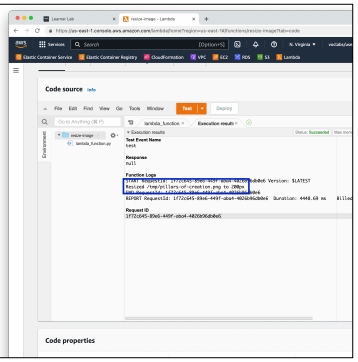
- Go back to the code view ar let's try our test again



AWS Lambda

Image Resizing in the Cloud

- No errors!
- We see our resize message.
- We have to copy our resized image somewhere
- Let's put it into an S3 bucket!
- Recommended to use different buckets for input and output to protect against recursive triggering of your function



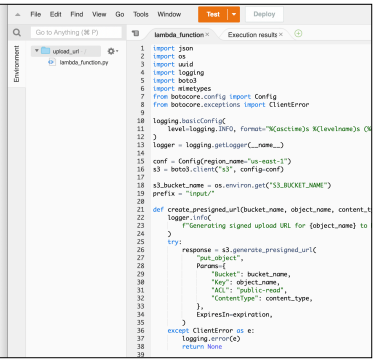
AWS Lambda S3 Demo

AWS Lambda
Upload Images

- How do we get our Chat client app to upload an image to our S3 bucket?
- AWS API calls!
- AWS S3 API provides a way to craft a 'signed' URL which we can use as the basis for a `PUT` or `POST` HTTP call to upload data directly to a bucket

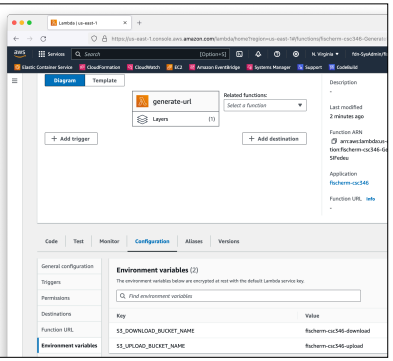
AWS Lambda
Upload Images

- Using the `boto3` SDK we can create an `s3_client` object and use the `generate_presigned_url` method
- Get the bucket name from an Environment Variable



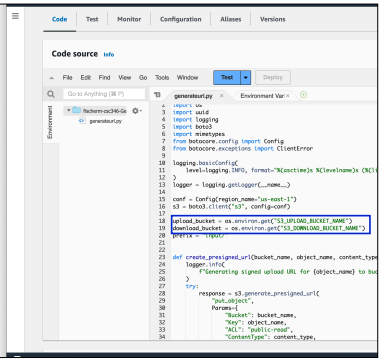
AWS Lambda
Environment Variables

- Just like almost every other code execution method, Lambda provides a way to define Environment Variables



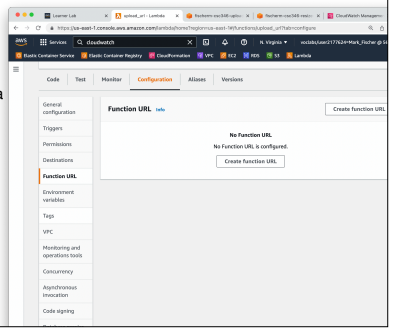
AWS Lambda
Environment Variables

- These are accessible from your code using standard language functions for accessing environment variables
- Code can be used in multiple runtime environments without having to know the specifics of the runtime



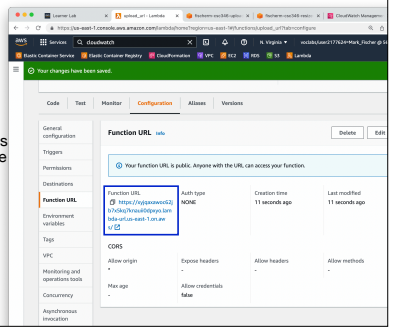
AWS Lambda
Function URLs

- For simple use cases, Lambda now provides a direct way to invoke the function through a URL
- Basic functionality
- API Gateway is a more robust and featured service for more production projects



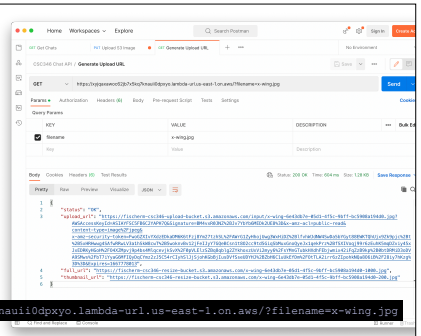
AWS Lambda
Function URLs

- For now we will not use any Authentication
- Potentially a security risk as this would allow anyone to generate upload URLs for our buckets and upload files
- Acceptable risk for now
- Could implement your own Basic Auth in the lambda function



AWS Lambda
Function URLs

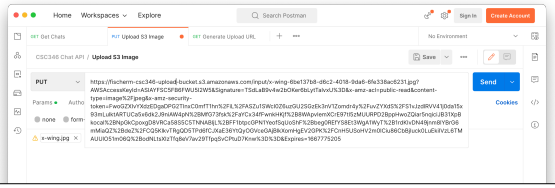
- Use our function URL in Postman
- GET
- Pass the file name in through query string parameters



```
https://xyjqaxawoc62jb7x5kq7knauii0dpxyo.lambda-url.us-east-1.on.aws/?filename=x-wing.jpg
```

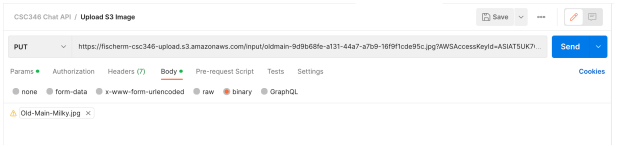

AWS Lambda
Function URLs

- Use the returned URL as the destination for a PUT HTTP request that passes a file



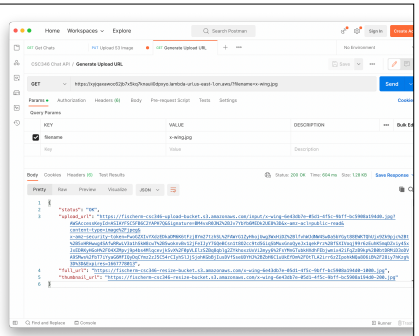
AWS Lambda
Function URLs

- In Postman, select the image in the Body tab, and choose “binary"



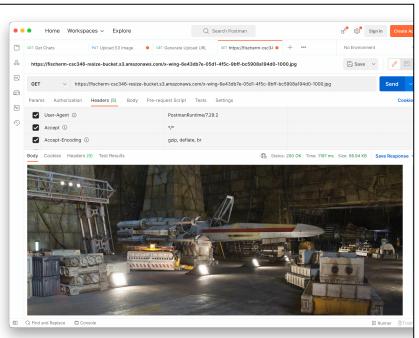
AWS Lambda
Function URLs

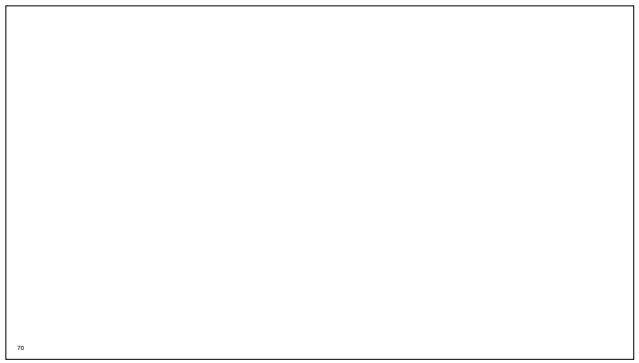
- Since we uploaded the image to the S3 bucket configured as the trigger for our resize function, the image should be resized automatically
- Our create upload URL call also returns the URLs of the resized objects
- We can view them directly



AWS Lambda
Function URLs

- Since we uploaded the image to the S3 bucket configured as the trigger for our resize function, the image should be resized automatically
- Our create upload URL call also returns the URLs of the resized objects
- We can view them directly





70