AWS Cloud Computing Project

A full-stack image uploader built on AWS using CLI-based deployment

AWS Cloud Computing Project

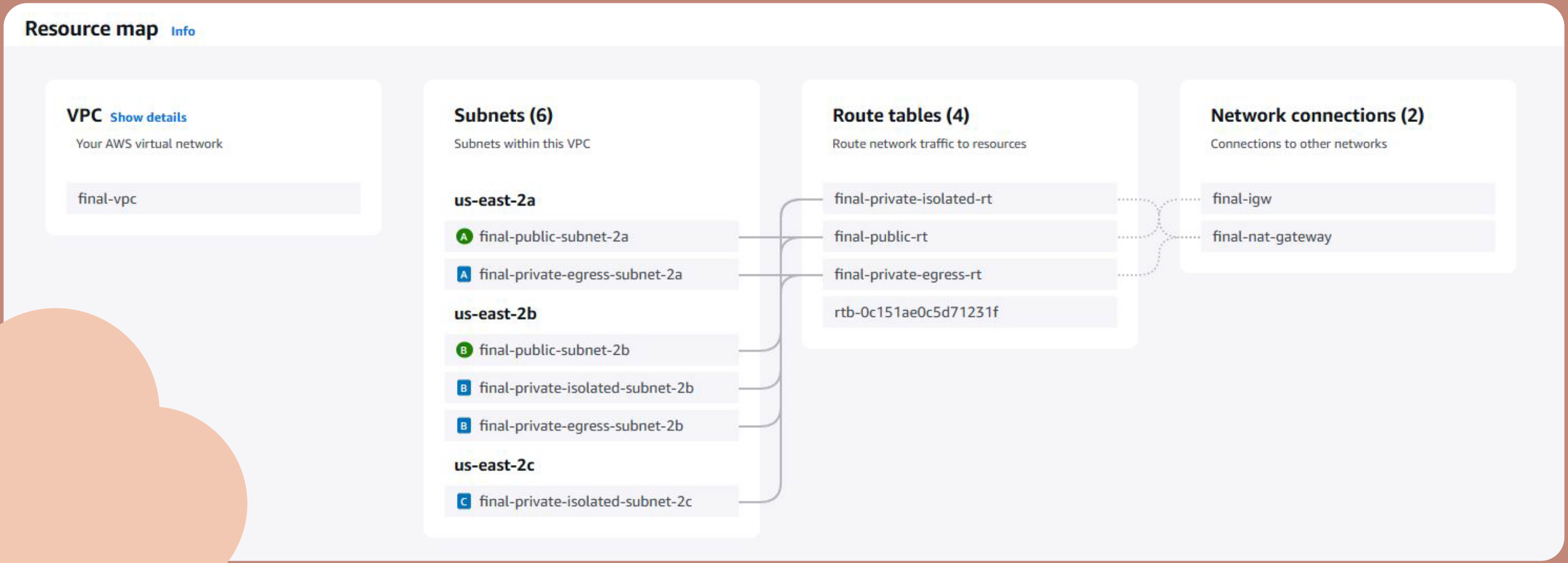
Overview

This project was developed as the final assignment for my Cloud Computing course and focused on deploying a full-stack web application using AWS infrastructure. The application allowed users to upload an image, which would then be automatically resized by an AWS Lambda function and displayed in a gallery of processed images. While the user-facing interface appeared simple, the backend architecture involved a custom VPC, multiple EC2 instances, S3 buckets, and serverless automation - all created using the AWS Command Line Interface (CLI).

Tech Stack: AWS CLI, EC2, S3, Lambda, Node.js, MongoDB, CloudWatch, IAM, Auto Scaling, Application Load Balancer.

Custom VPC & Subnet Configuration

To begin, I created a custom VPC with multiple subnets across different availability zones. This architecture was essential for managing network traffic (ingress/egress) and improving resilience in case of an outage. The subnets were classified into public, private egress, and private isolated zones to reflect the level of access needed by each component.



EC2 Instances & Security

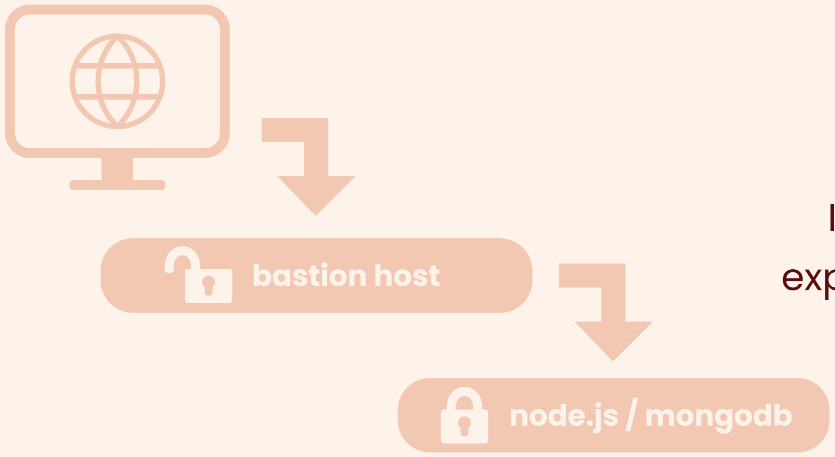
I launched three EC2 instances:

A Node.js backend server

A MongoDB database instance

A bastion host

for secure SSH access into private instances



Managing access between instances proved tricky. I spent significant time configuring security groups and IAM roles to ensure the correct ports were open without exposing sensitive resources. I had to SSH into the bastion host and then tunnel into the private instances – this layering added security but also complexity.

S3 Buckets & Lambda Function

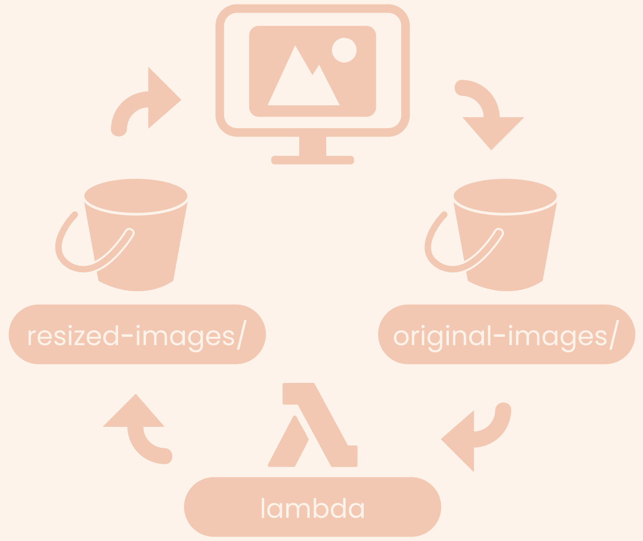
The image processing workflow used two S3 buckets:

original-images/

where images were uploaded

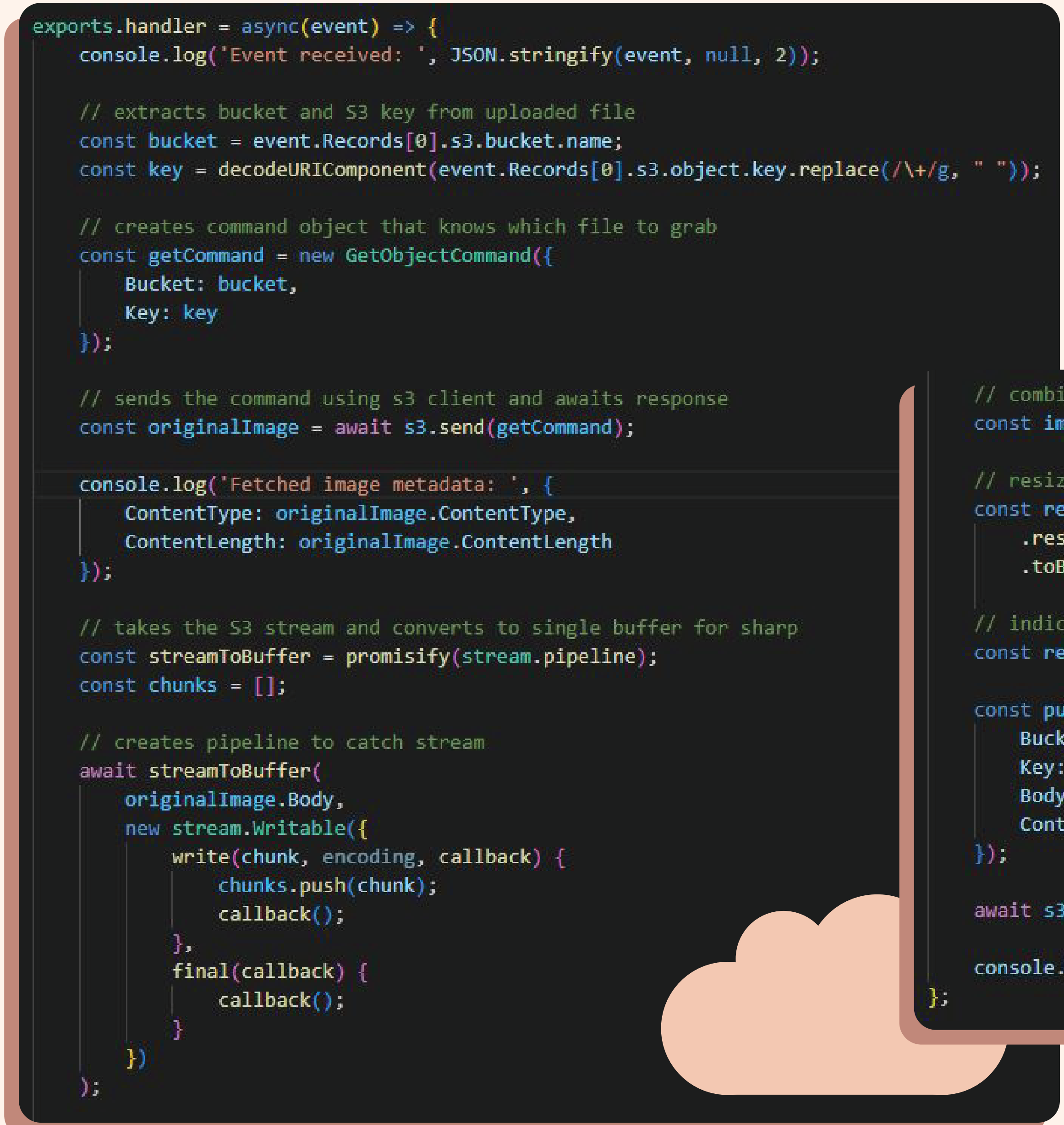
resized-images/

where Lambda saved the processed 300px-wide images

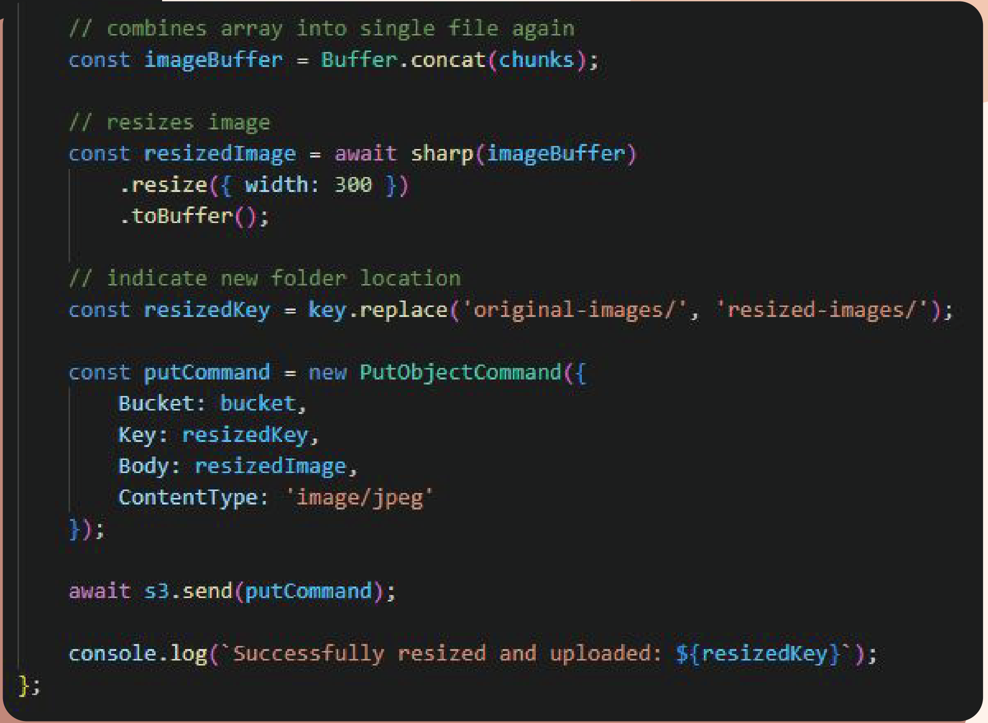


The Lambda function was triggered upon image upload. It resized the image using the Sharp library and saved the result in the second bucket. Setting up the event trigger and getting the function to process images correctly was one of the most technical parts of the project.

snippet of Lambda function



```
exports.handler = async(event) => {
    console.log('Event received: ', JSON.stringify(event, null, 2));

    // extracts bucket and S3 key from uploaded file
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));

    // creates command object that knows which file to grab
    const getCommand = new GetObjectCommand({
        Bucket: bucket,
        Key: key
    });

    // sends the command using s3 client and awaits response
    const originalImage = await s3.send(getCommand);

    console.log('Fetched image metadata: ', {
        ContentType: originalImage.ContentType,
        ContentLength: originalImage.ContentLength
    });

    // takes the S3 stream and converts to single buffer for sharp
    const streamToBuffer = promisify(stream.pipeline);
    const chunks = [];

    // creates pipeline to catch stream
    await streamToBuffer(
        originalImage.Body,
        new stream.Writable({
            write(chunk, encoding, callback) {
                chunks.push(chunk);
                callback();
            },
            final(callback) {
                callback();
            }
        })
    );
```



```
    // combines array into single file again
    const imageBuffer = Buffer.concat(chunks);

    // resizes image
    const resizedImage = await sharp(imageBuffer)
        .resize({ width: 300 })
        .toBuffer();

    // indicate new folder location
    const resizedKey = key.replace('original-images/', 'resized-images/');

    const putCommand = new PutObjectCommand({
        Bucket: bucket,
        Key: resizedKey,
        Body: resizedImage,
        ContentType: 'image/jpeg'
    });

    await s3.send(putCommand);

    console.log(`Successfully resized and uploaded: ${resizedKey}`);
};
```

Testing & Validation

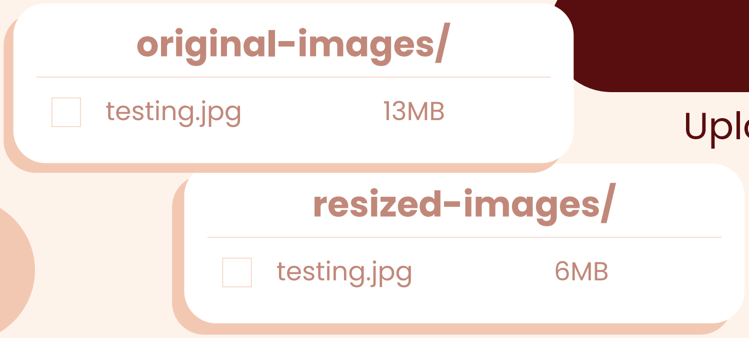
I tested the app in three key phases:

SSH & App Boot Up

Verified the app ran by SSH-ing into the backend instance and confirming the Node server was active.

```
> Server running on host http://localhost:8080/
```

Lambda Trigger



original-images/

☐	testing.jpg	13MB

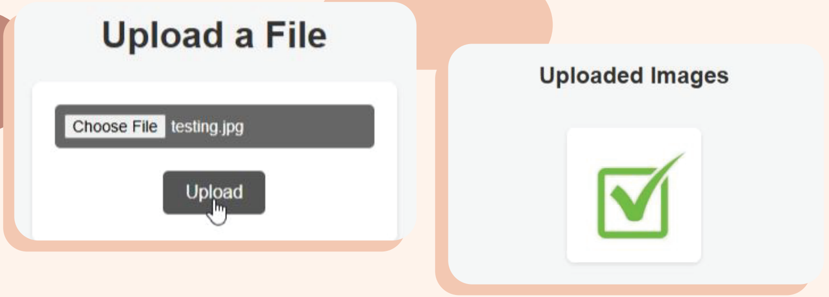
resized-images/

☐	testing.jpg	6MB

Uploaded an image manually to S3, then checked the resized version appeared in the correct bucket.

Full App Flow

Uploaded an image through the frontend UI and confirmed it appeared in the app's image gallery.



Challenges & Roadblocks

Region Confusion

I accidentally created resources in both us-east-1 and us-east-2, which broke communication between services. Recreating those resources in the correct region taught me to always double-check the AWS region.

EC2 Instances

us-east-1

No Results Found

Lambda Upload Errors

Failed to upload package

My Lambda zip package was built on a non-Linux system, which caused failures. I had to repackage the files using Docker to convert the files and ensure compatibility.

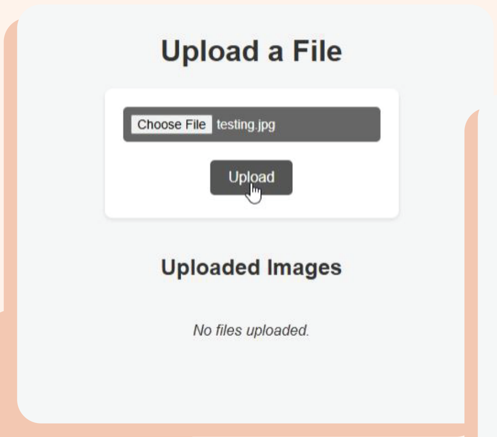
SSH / Instance Access

Multiple security layers meant even one misconfigured rule could block access. Fixing this involved balancing port access, IAM permissions, and VPC routing.

```
$ ssh -i a2-project-pair.pem ubuntu@18.220.54.63
  ssh: connect to host 18.220.54.63 port 22: Commection timed out
```

Results & Reflection

The project successfully achieved its goal: creating a secure, scalable, fully-deployed full-stack application using AWS. Users could upload an image, have it processed by a Lambda function, and view the resized image in the app's interface. But beyond the surface-level functionality, this project pushed me to develop key skills in cloud architecture, networking, and CLI-based deployments.

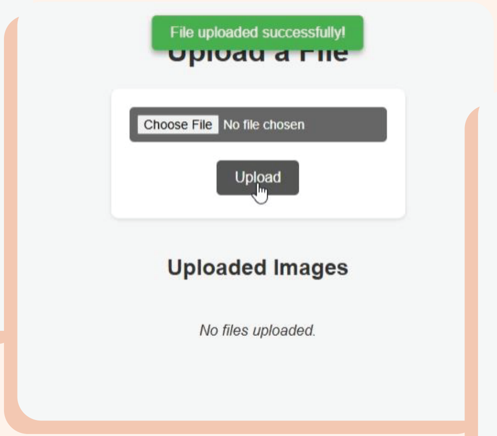

Upload a File

Choose File testing.jpg

Upload

Uploaded Images

No files uploaded.


File uploaded successfully!

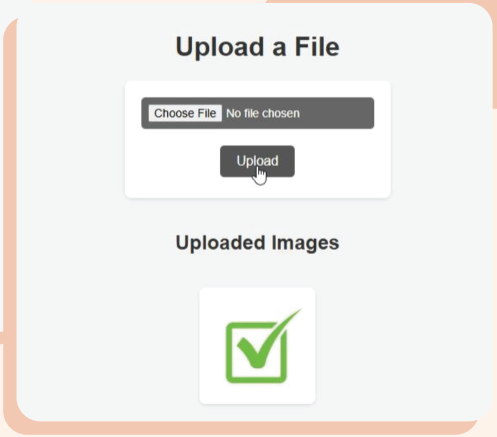
Upload a File

Choose File No file chosen

Upload

Uploaded Images

No files uploaded.


Upload a File

Choose File No file chosen

Upload

Uploaded Images

Wins

I overcame a difficult Lambda upload issue that even stumped my mentor. Solving it independently was a major personal win.

I now feel confident navigating AWS CLI and deploying real-world infrastructure from scratch.

Lessons Learned

Always track the region you're operating in.
A single mismatch can break everything.

Make notes of commonly used CLI commands early on.
Repetition and reference saved me time as the project grew.

I realized how valuable visual planning is - even when working in a non-visual environment like CLI.

Final Thoughts

What looked like a simple image uploader app was actually a powerful learning experience in full-stack cloud deployment. Every piece of the architecture - from Lambda automation to subnet routing - was deliberately planned, tested, and iterated on. This project gave me the confidence to say that not only can I build cloud-based applications - I can troubleshoot, secure, and scale them too.