

## 1. Backend: DynamoDB, API Gateway, Lambda function, SNS

The screenshot shows the AWS Management Console for the 'ImageUploads' table in DynamoDB. The left sidebar contains navigation links for 'DynamoDB' and 'DAX'. The main content area displays the 'ImageUploads' table details. A notification banner at the top indicates that Point-in-time recovery (PITR) is enabled. Below this, the 'General information' section provides details about the table's configuration.

General information			
Partition key email (String)	Sort key -	Capacity mode On-demand	Table status <span>Active</span>
Alarms <span>No active alarms</span>	Point-in-time recovery (PITR) <span>Off</span>	Item count 0	Table size 0 bytes
Average item size 0 bytes	Resource-based policy <span>Not active</span>		
Amazon Resource Name (ARN) <code>arn:aws:dynamodb:us-east-1:103619411293:table/imageUploads</code>			
<a href="#">Additional info</a>			

The screenshot shows the AWS Management Console for the 'uploadImageToS3' resource in API Gateway. The left sidebar contains navigation links for 'API Gateway' and 'Resources'. The main content area displays the 'uploadImageToS3' resource details. A notification banner at the top indicates that CORS is successfully enabled. Below this, the 'Resources' section shows the 'POST' method configuration.

**Resources**

**/ - POST - Method execution**

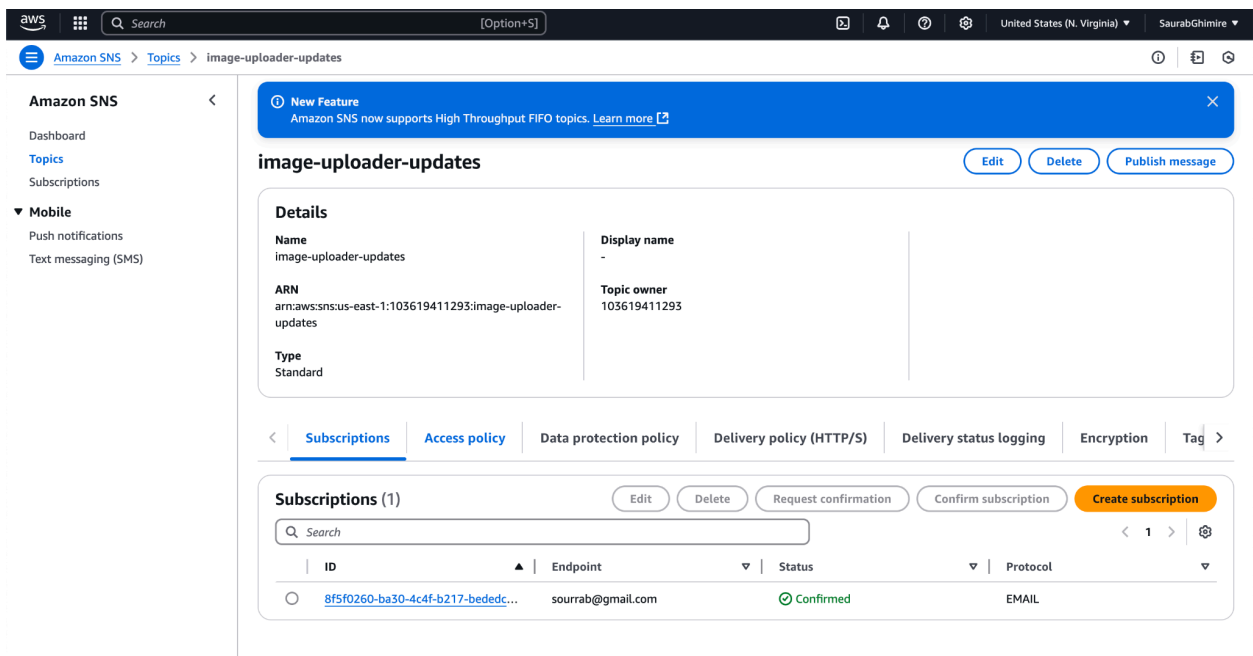
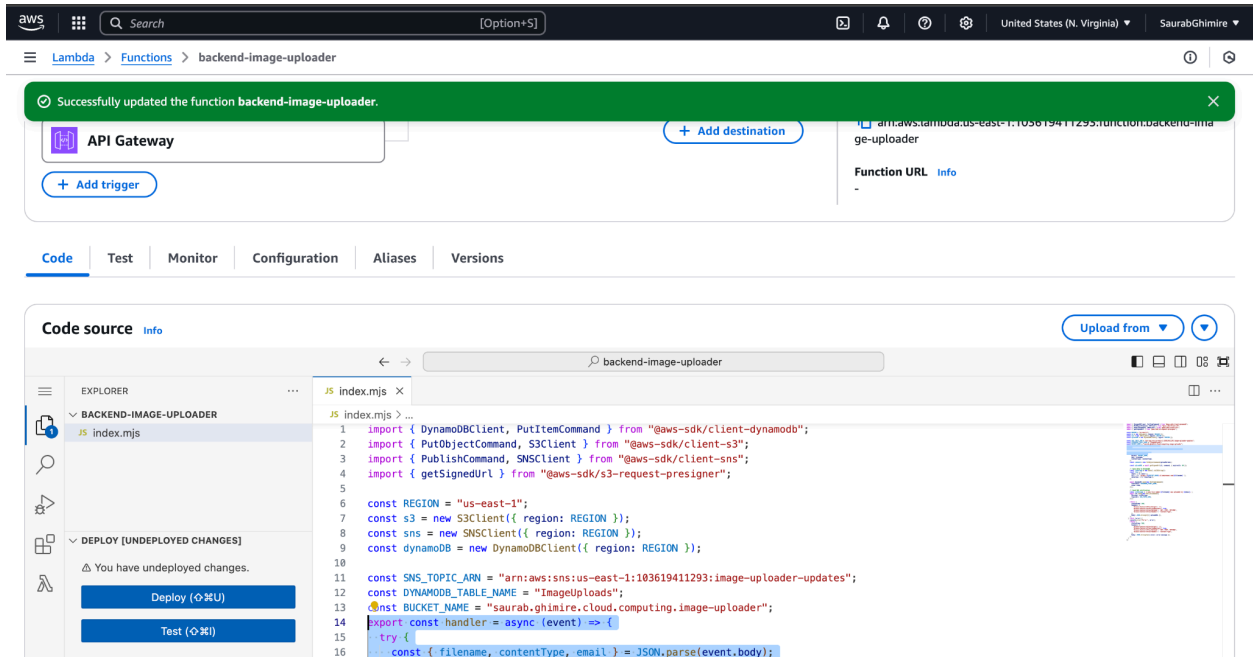
ARN: `arn:aws:execute-api:us-east-1:103619411293:0pg58xyup9/*/*POST/` Resource ID: `6qmagdt18j`

The diagram illustrates the method execution flow:

```
graph LR; Client[Client] --> MR[Method request]; MR --> IR[Integration request]; IR --> LI[Lambda integration]; LI --> IRR[Integration response Proxy integration]; IRR --> MR2[Method response]; MR2 --> Client
```

**Method request settings**

Authorization	API key required
NONE	False



2. Frontend: S3 for React, CloudFront, S3 for images

3. Deployed website: Show the live website with working image upload functionality.