

## Create a S3 Bucket:

The screenshot shows the AWS S3 console interface. On the left, a sidebar menu includes 'Buckets', 'Access management and security', 'Storage management and insights', and 'Account and organization settings'. The main content area displays 'General purpose buckets (1)'. A table lists one bucket: 'image-resize-demo-shail' (Name), 'Asia Pacific (Mumbai) ap-south-1' (AWS Region), and 'December 19, 2025, 20:56:02 (UTC+05:30)' (Creation date). Action buttons include 'Copy ARN', 'Empty', 'Delete', and 'Create bucket'. To the right, there are sections for 'Account snapshot' (updated daily) and 'External access summary - new' (info).

## Create a Lambda Function :

The screenshot shows the AWS Lambda console interface. On the left, a sidebar menu includes 'Functions', 'Additional resources', and 'Related AWS resources'. The main content area displays 'Functions (1)'. A table lists one function: 'imageResizeFunction' (Function name), '-' (Description), 'Zip' (Package type), 'Node.js 20.x' (Runtime), 'Standard' (Type), and '7 minutes ago' (Last modified). Action buttons include 'Actions' and 'Create function'. To the right, there is a 'Tutorials' section titled 'Create a simple web app' with a description and a 'Start tutorial' button.

## First Make a Layer using the CloudShell:

The screenshot shows the AWS Lambda Layers page. On the left, there's a sidebar with navigation links like Dashboard, Applications, Functions, Additional resources (Capacity providers, Code signing configurations, Event source mappings, Layers, Replicas), and Related AWS resources (Step Functions state machines). The main area displays a table titled "Layers (1)". The table has columns: Name, Version, Description, Compatible runtimes, Compatible architectures, and Created. One row is shown for "sharp-layer" version 1, which was created 24 minutes ago. The "Created" column shows a timestamp of "24 minutes ago". On the right side of the page, there's a "Tutorials" section with a link to "Create a simple web app".

This screenshot shows the detailed view of the "sharp-layer" layer. It includes a "Version details" section with fields for Version (1), Version ARN (arn:aws:lambda:ap-south-1:867852670365:layer:sharp-layer:1), Description (-), Created (24 minutes ago), License (-), and Compatible runtimes (nodejs20.x). Below this, there are tabs for "Versions" and "Functions using this version". Under "Versions", it says "All versions (1)" and shows a table with one entry: Version 1, Version ARN arn:aws:lambda:ap-south-1:867852670365:layer:sharp-layer:1, and Description (-). On the right, there's a "Tutorials" section with a link to "Create a simple web app".

## Add a Layer:

The screenshot shows the 'Layers' section of the AWS Lambda console. A single layer named 'sharp-layer' is listed with the following details:

Merge order	Name	Layer version	Compatible runtimes	Compatible architectures	Version ARN
1	sharp-layer	1	nodejs20.x	x86_64	arn:aws:lambda:ap-south-1:867852670365:layer:sharp-layer:1

Buttons for 'Edit' and 'Add a layer' are visible at the top right.

## Make a API Gateway :

The screenshot shows the 'APIs' list page in the AWS API Gateway console. One API named 'image-resize-api' is listed:

Name	Description	ID	Protocol	API endpoint type	Created	Security policy	API status
image-resize-api		gmrai8uxkf	REST	Regional	2025-12-19	SecurityPolicy_TLS13_1_3_FIPS_2025_09	Available

The screenshot shows the 'Resources' page for the 'image-resize-api' API in the AWS API Gateway console. It displays a single resource path '/resize' with methods 'OPTIONS' and 'POST' defined.

**Resource details:**

- Path: /
- Resource ID: 7djj6isv7c
- Update documentation | Enable CORS

**Methods (0):**

Method type	Integration type	Authorization	API key
No methods			

The screenshot shows the AWS API Gateway Stages configuration page. The left sidebar shows the API Gateway navigation bar and the selected API: 'image-resize-api'. Under the 'Stages' section, 'prod' is selected. The main area displays 'Stage details' for 'prod', showing a Rate of 10000 and a Burst of 5000. It also shows 'Logs and tracing' settings, both of which are currently inactive.

## Index.html :

```
<!DOCTYPE html>

<html>
  <head>
    <title>AWS Image Resizer</title>
  </head>
  <body>
    <h2>Upload Image</h2>
    <input type="file" id="file" />
    <button onclick="upload()">Upload & Resize</button>

    <p id="status"></p>
    <a id="download" target="_blank"></a>
```

```
<script>

async function upload() {

    const file = document.getElementById("file").files[0];
    const reader = new FileReader();

    reader.onloadend = async () => {

        const base64 = reader.result.split(",")[1];

        const res = await fetch("https://gmrai8uxkf.execute-api.ap-south-1.amazonaws.com/prod/resize", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ image: base64 }),
        });

        const data = await res.json();

        document.getElementById("status").innerText = data.message;
        document.getElementById("download").href = data.downloadUrl;
        document.getElementById("download").innerText = "Download Resized Image";
    };

    reader.readAsDataURL(file);
}

</script>
</body>
</html>
```

## Lamda Function :

```
import {  
    S3Client,  
    PutObjectCommand,  
    GetObjectCommand  
} from "@aws-sdk/client-s3";  
  
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";  
  
import sharp from "sharp";  
  
import { randomUUID } from "crypto";  
  
  
  
const s3 = new S3Client({ region: process.env.AWS_REGION });  
const BUCKET_NAME = "image-resize-demo-shail";  
  
  
  
export const handler = async (event) => {  
    try {  
        const body = JSON.parse(event.body);  
        const imageBuffer = Buffer.from(body.image, "base64");  
  
  
  
        const originalFileName = `original-${randomUUID()}.jpg`;  
        const resizedFileName = `resized-${randomUUID()}.jpg`;  
    } catch (error) {  
        console.error(error);  
    }  
};
```

```
// 1 Upload ORIGINAL image to uploads/
await s3.send(
  new PutObjectCommand({
    Bucket: BUCKET_NAME,
    Key: `uploads/${originalFileName}`,
    Body: imageBuffer,
    ContentType: "image/jpeg",
  })
);
```

```
// 2 Resize image
const resizedImage = await sharp(imageBuffer)
  .resize(300, 300)
  .jpeg()
  .toBuffer();
```

```
// 3 Upload RESIZED image to resized/
await s3.send(
  new PutObjectCommand({
    Bucket: BUCKET_NAME,
```

```
    Key: `resized/${resizedFileName}`,
    Body: resizedImage,
    ContentType: "image/jpeg",
  })
};

// 4 Generate presigned URL for resized image
const getCommand = new GetObjectCommand({
  Bucket: BUCKET_NAME,
  Key: `resized/${resizedFileName}`,
});
}

const signedUrl = await getSignedUrl(s3, getCommand, {
  expiresIn: 3600, // 1 hour
});

return {
  statusCode: 200,
  headers: {
    "Access-Control-Allow-Origin": "*",
  },
}
```

```
body: JSON.stringify({
    message: "Image uploaded and resized successfully",
    downloadUrl: signedUrl,
}),
};

} catch (error) {
    console.error("ERROR:", error);
    return {
        statusCode: 500,
        headers: {
            "Access-Control-Allow-Origin": "*",
        },
        body: JSON.stringify({ error: error.message }),
    };
}
};
```

