LAB DA 5

1. CLOUDTRAIL:

Step 1: Create a Trail
Open the **AWS Management Console → CloudTrail → Trails → Create trail**.
Enter the trail name **"MySecurityTrail"** and apply it to **all regions**.
Disable Log file SSE KMS encryption

Step 2: Configure Storage
Choose **Create a new S3 bucket** to store CloudTrail logs.
Use the default settings or the suggested bucket name.
Enable **Log file validation and CloudWatch Logs integration** for real-time monitoring.

Next page
Step 3: Enable Event Logging
Enable **Management events** and select both **Read** and **Write**.
Click **Next**, review your configuration, and then click **Create trail**.
Your trail will now begin logging activities.

Step 4: Generate Some Activity
Perform a few simple AWS operations, such as:

- Launch or stop a **t2.micro EC2 instance**.
- Create or delete an **S3 bucket**.
  These actions will generate API logs recorded by CloudTrail.

Step 5: View and Analyze Events
Go to **CloudTrail → Event history** after a few minutes.
You'll see logs of your recent activities.
Open any event to check:

- **Who performed the action (IAM user or role)**
- **When it happened (timestamp)**
- **Source IP address**
- **Resource affected**

Step 6: Record Results
Take a **screenshot** of the Event history showing recent events. Then click on createKeyPair- Open one event and copy the **JSON event record** into your report.
Highlight details like eventName, userIdentity, sourceIPAddress, and eventTime.

Step 7: Conclusion
AWS CloudTrail continuously logs all API actions in your account.
It helps track **who did what, when, and from where**, making it crucial for **auditing**, **incident investigation**, and **security monitoring**.

AWS CloudTrail is a monitoring and auditing service that records every API call and activity made within an AWS account. It tracks important details such as the identity of the user or role making the request, the time of the action, the source IP address, and the resources affected.
This detailed logging helps with **auditing** by providing a chronological record of all operations performed across AWS services, allowing administrators to verify compliance, monitor changes, and ensure accountability.
In terms of **incident investigation**, CloudTrail is invaluable during security analysis or troubleshooting. It enables security teams to trace unauthorized or suspicious actions, identify who made the change, and understand the context of the event. By reviewing CloudTrail logs, investigators can quickly pinpoint the root cause of incidents, mitigate risks, and strengthen security controls.

2. CLOUDWATCH:

Step 1: Launch an EC2 Instance
Start by launching a **t2.micro EC2 instance** running **Amazon Linux 2**. This instance will be the resource monitored through CloudWatch.

NAME : cw-agent-lab; amazo, t3, keypair generate
Go to advanced details: create IAM instance profile (create role ->aws service-> usecase:ec2-> search for CloudWatchAgentServerPolicy and select it-> Name it CloudWatchAgentRole-> create role-> go back to ec2 and select it in iam instance profile
Now launch instance and connect->youll get a bird kinda black screen

sudo yum -y update
sudo yum install amazon-cloudwatch-agent -y
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
options:
On which OS are you planning to use the agent?
1. linux
2. windows
3. darwin
default choice: [1]:
1

Trying to fetch the default region based on ec2 metadata...
I! imds retry client will retry 1 timesAre you using EC2 or On-Premises hosts?
1. EC2
2. On-Premises

default choice: [1]:
1
Which user are you planning to run the agent?
1. cwagent
2. root
3. others
default choice: [1]:
2

Do you want to turn on StatsD daemon?
1. yes
2. no
default choice: [1]:
2

Do you want to monitor metrics from CollectD? WARNING: CollectD must be installed or the Agent will fail to start
1. yes
2. no
default choice: [1]:
2

Do you want to monitor any host metrics? e.g. CPU, memory, etc.
1. yes
2. no
default choice: [1]:
1

Do you want to monitor cpu metrics per core?
1. yes
2. no
default choice: [1]:
1

Do you want to add ec2 dimensions (ImageId, InstanceId, InstanceType, AutoScalingGroupName) into all of your metrics if the info is available?
1. yes
2. no
default choice: [1]:
1

Do you want to aggregate ec2 dimensions (InstanceId)?
1. yes
2. no

default choice: [1]:
1
Would you like to collect your metrics at high resolution (sub-minute resolution)? This enables sub-minute resolution for all metrics, but you can customize for specific metrics in the output json file.
1. 1s
2. 10s
3. 30s
4. 60s
default choice: [4]:
4

Which default metrics config do you want?
1. Basic
2. Standard
3. Advanced
4. None
default choice: [1]:
1

Are you satisfied with the above config? Note: it can be manually customized after the wizard completes to add additional items.
1. yes
2. no
default choice: [1]:
1

Do you have any existing CloudWatch Log Agent (http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AgentReference.html) configuration file to import for migration?
1. yes
2. no
default choice: [2]:
2

Do you want to monitor any log files?
1. yes
2. no
default choice: [1]:
2

Do you want the CloudWatch agent to also retrieve X-ray traces?
1. yes
2. no

default choice: [1]:
2
Do you want to store the config in the SSM parameter store?
1. yes
2. no
default choice: [1]:
2
Program exits now.

sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json -s

Step 2: Install and Configure the CloudWatch Agent
Install and set up the **CloudWatch Agent** to collect detailed system metrics. This agent allows CloudWatch to gather more information beyond default metrics, such as memory and disk performance data, for deeper visibility.

Go to cloudwatch->all metrics->browse->ec2->per instance metrics->search for cw-agent-lab and take ss of graphs

Step 3: Select Metrics to Monitor -NA
During the setup process, choose key parameters to monitor for performance evaluation:

- **CPU Utilization** – Measures the instance's processor load.
- **Memory Usage** – Tracks memory consumption.
- **Disk I/O** – Monitors input/output activity on attached storage.

These metrics help understand how efficiently the instance is operating.

Step 4: Start the CloudWatch Agent - NA
Activate the agent so it begins transmitting the selected metrics to the CloudWatch console. Once running, it continuously sends performance data that can be viewed and analyzed in real time.

Step 5: Verify Metrics in CloudWatch Console -NA
Go to the **AWS Console → CloudWatch → Metrics → EC2** and check that the data is being displayed correctly. You should see live graphs showing CPU, memory, and disk activity from your instance.

Step 6: Create a CloudWatch Alarm
CHOOSE ALL ALARMS OPTION-> create alarm
Select metric

- click "Select metric"
- choose EC2 → Per-Instance Metrics

- click your instance → check CPUUtilization
- click Select metric

then change period to 1 min; conditions GREATER than 70
Additional configurations-> 2 out of 2
Configure actions-> In alarm-> create new topic->enter name and gmail->next->name the alarm to HighCPU-Usage-alarm->next->create->click on it and take ss of the graph

Set up an alarm to automatically alert you when the CPU usage is too high.

- **Metric:** CPUUtilization
- **Condition:** Greater than 70% for 2 consecutive periods
- **Notification:** Use an **SNS topic** to send an email alert.

This ensures timely awareness of performance spikes or abnormal behavior.

Finally, take ss of the gmail

LAB 2

3. ELASTIC BEANSTALK

Step 1: Log in to the **AWS Management Console**, search for **Elastic Beanstalk**, and click **Create application**.

Step 2: Enter an **Application name** (for example: shoelab). Leave defaults for other fields.

Step 3: Under **Environment tier**, choose **Web server environment**.

Step 4: Under **Platform**, choose **Python**. Then select the latest **Python 3.x on 64-bit Amazon Linux 2** branch.

Step 5: Under **Application code**, choose **Upload your code**. Upload the prepared ZIP file shoestore_flask_flat_full.zip and type a **Version label** (for example: v1).

Step 6: Click **Next**. On the **Service access** page, click **Create role** for both **Service role** and **EC2 instance profile**. Once they are created, select them from the dropdowns.

Step 7: Click **Next** through **Networking, Database, and Tags**. Do not change anything (leave defaults).

Step 8: Click **Next** through **Instance traffic, scaling, monitoring, and logging**. Leave all settings as default (single instance, free tier eligible).

Step 9: On the **Review** page, check your settings and then click **Create environment**. AWS will now create the environment, EC2 instance, and load balancer. This takes several minutes.

Step 10: When the environment status shows **Health: Green**, copy and open the **environment URL** shown at the top (it looks like http://something.elasticbeanstalk.com). This loads your Flask shoe store.

Step 11: In your environment, go to **Configuration → Software → Edit**. Under **Environment properties**, add a new variable:

- Key: SECRET_KEY
- Value: any long random string (for Flask sessions).
  Click **Apply**. The app restarts automatically.

Step 12: Verify the deployment by visiting:

- /products → product catalog loads.
- /cart → add and view items.
- /checkout → test the demo order form.
- /health → returns {"ok": true}.

Step 13: After testing, clean up to avoid charges. In Elastic Beanstalk, open your environment → **Actions → Terminate environment**. Confirm by typing the environment name.

Beanstalk->create application->web server environment-> fill application name and env ->node.js->upload your code ->local file->use domain to print

LAB 1:

4. Migrate and SQL

: **Launch a Python Application on EC2 in One Region**

Step 1: Log in to AWS

Step 2: Launch EC2 Instance in a Region (e.g., N. Virginia)

1. Go to **EC2 > Instances > Launch Instances**.
2. Name the instance: PythonAppServer.
3. Choose Amazon Machine Image (AMI): **Ubuntu Server 22.04 LTS**.
4. Choose Instance Type: t2.micro (free tier).
5. Key Pair: Create new or use existing (download .pem file).
6. Configure:
   - Allow **SSH (22)** and **HTTP (80)** in security group.
7. Launch the instance.

Step 3: Connect via SSH

Click on connect

sudo apt update

sudo apt install python3 -y

echo "print('Hello World')" > app.py

python3 app.py

part b)

Migrate EC2 Instance to Another Region

Direct migration is not allowed. Use AMI + Snapshots.

Step 1: Create an AMI

- Go to EC2 > Instances > Select instance > **Actions > Image > Create Image**
- Name it and create.

Step 2: Copy AMI to Another Region (e.g., Mumbai)

- Go to **EC2 > AMIs > Select AMI > Actions > Copy AMI**
- Choose destination region and copy.

Step 3: Launch Instance in New Region

- Switch to new region (top-right corner).

Go to **AMIs**, find the copied AMI, launch a new instance

SQL:

Part c)

Setup Another EC2 for MySQL and Basic DB Operations

Step 1: Launch Another EC2 Instance (e.g., Name: MySQLServer)

- Repeat EC2 launch steps as above.
- Allow SSH & MySQL (port 3306) in security group.

Step 2: Connect and Install MySQL

Step 3: Login and Create Database

```
sudo apt update
sudo apt install mysql-server
sudo mysql
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password by 'password';
FLUSH PRIVILEGES;

-- Create the main database
CREATE DATABASE college_db;
USE college_db;

-- Create courses table
CREATE TABLE courses (
  course_id INT PRIMARY KEY,
  course_name VARCHAR(50),
  department VARCHAR(50),
  credits INT
);

-- Insert data into courses
INSERT INTO courses VALUES
(101, 'Data Structures', 'Computer Science', 4),
(102, 'Microeconomics', 'Business', 3),
(103, 'Linear Algebra', 'Mathematics', 4);

-- Create faculty table
CREATE TABLE faculty (
  faculty_id INT PRIMARY KEY,
  faculty_name VARCHAR(50),
  department VARCHAR(50),
  salary DECIMAL(10,2)
);
```

```
-- Insert data into faculty
INSERT INTO faculty VALUES
(1, 'Dr. Sharma', 'Computer Science', 75000),
(2, 'Prof. Mehta', 'Mathematics', 68000),
(3, 'Dr. Rao', 'Business', 72000);



-- 1. View data
SELECT * FROM courses;
SELECT * FROM faculty;

-- 2. INNER JOIN
SELECT f.faculty_name, c.course_name, f.department
FROM faculty f
INNER JOIN courses c ON f.department = c.department;

-- 3. Average salary
SELECT AVG(salary) AS avg_salary FROM faculty;

-- 4. Count of courses
SELECT COUNT(*) AS total_courses FROM courses;

-- 5. Highest salary
SELECT faculty_name, salary FROM faculty ORDER BY salary DESC LIMIT 1;

-- 6. Total credits offered
SELECT SUM(credits) AS total_credits FROM courses;

-- 7. Department with multiple courses
```

```
SELECT department, COUNT(course_id) AS num_courses
FROM courses
GROUP BY department;

-- 8. Faculty per department
SELECT department, COUNT(faculty_id) AS num_faculty
FROM faculty
GROUP BY department;

-- 9. Average salary per department
SELECT department, AVG(salary) AS avg_salary
FROM faculty
GROUP BY department;

-- 10. Courses with >3 credits
SELECT course_name FROM courses WHERE credits > 3;

EXIT;
mysqldump -u root -p college_db > college_db_backup.sql
# Enter password: password

ls -l college_db_backup.sql
```

Once you've finished creating tables and inserting data, just **run this command in your EC2 terminal (not inside MySQL):**

```
mysqldump -u root -p college_db > college_db_backup.sql
```

5. S3 BUCKET IN AWS – STATIC HOSTING

Create S3 Bucket: Go to S3 → Create a globally unique bucket name → Uncheck "Block all public access" -> create bucket -> click on it ->UPLOAD FILESSSS

Upload Files: Upload your index.html, styles.css, script.js, etc., to the bucket.

Enable Static Website Hosting: click on the bucket->In **Properties**, enable static website hosting → Set index.html as index document.

Make Bucket Public: In **Permissions** → Add a bucket policy to allow public read access

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::demobuck123555/*"
  }
 ]
}
```

Enable Versioning: In **Properties** → Enable bucket versioning to keep file history.

Access Website: Use the **S3 website endpoint URL** to view your live site.

Index1.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>About Me</title>
  <style>
    body {
      font-family: 'Poppins', sans-serif;
      margin: 0;
      padding: 0;
      background: linear-gradient(to right, #f5f5dc, #d2b48c);
      color: #5d4037;
    }
    header {
      background: #8b4513;
      color: white;
```

```
    text-align: center;
    padding: 20px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
}
marquee {
    font-size: 18px;
    color: #fff;
    background-color: #d2691e;
    padding: 10px 0;
    margin: 0;
}
section {
    padding: 20px;
    margin: 20px auto;
    background-color: rgba(255, 248, 220, 0.9);
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    max-width: 800px;
}
h2 {
    color: #8b4513;
    margin-bottom: 10px;
}
img {
    max-width: 100%;
    border-radius: 10px;
}
.content {
    display: flex;
    gap: 20px;
    flex-wrap: wrap;
}
.content div {
    flex: 1;
    min-width: 250px;
}
.image-section {
    text-align: center;
}
.image-section img {
    width: 150px;
    border-radius: 50%;
    margin: 10px auto;
}
footer {
    text-align: center;
    padding: 15px;
    background: #8b4513;
    color: white;
    position: static;
    margin-top: 20px;
}
.social-links a {
    text-decoration: none;
    margin: 0 10px;
    color: #ffdab9;
    font-size: 18px;
}
```

```
    .social-links a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <header>
    <h1>Welcome to my first webpage!</h1>
  </header>

  <marquee behavior="scroll" direction="left">Hi, I'm Keerti Damani, and this is a glimpse into my life, passions, and projects!</marquee>

  <section class="image-section">
    <h2>Hello, I'm Keerti! (22BBS0174)</h2>
    <p>A tech enthusiast, programmer, and lifelong learner.</p>
  </section>

  <section>
    <h2>Introduction</h2>
    <p>I am a B.Tech student at VIT Vellore, specializing in Computer Science with a focus on Business Systems. My journey is fueled by a passion for technology and innovation, constantly seeking to explore new horizons in the tech world.</p>
  </section>

  <section>
    <h2>My Skills</h2>
    <div class="content">
      <div>
        <h3>Programming Languages</h3>
        <ul>
          <li>Python</li>
          <li>C, C++</li>
          <li>Java</li>
        </ul>
      </div>
      <div>
        <h3>Other Skills</h3>
        <ul>
          <li>Web Development</li>
          <li>Data Science</li>
          <li>SuiteScripts</li>
        </ul>
      </div>
    </div>
  </section>

  <section>
    <h2>My Interests</h2>
    <p>I enjoy exploring emerging technologies, contributing to tech communities, and being part of exciting projects. As Vice Chair of the French Literary Association, I actively contribute to cultural events that bring students together.</p>
    <img src="tech_image.png" alt="Interest-related image">
  </section>

  <section>
    <h2>Fun Facts</h2>
```

```
    <p>📚 I love reading books on AI and ML.<br>🌍 I'm learning French and exploring diverse cultures.<br>🎨 In my downtime, I enjoy designing interfaces and working on creative projects.</p>
  </section>

  <footer>
    <p>Contact me: <a href="mailto:keertidamani@example.com">keertidamani@example.com</a></p>
    <div class="social-links">
      <a href="https://in.linkedin.com/in/keertidamani">LinkedIn</a> | <a href="https://github.com/keertidamani">GitHub</a> | <a href="#">Portfolio</a>
    </div>
  </footer>
</body>
</html>
```

6. LAB 4:

A company wants to deploy a secure, scalable, and highly available web application on AWS for global users. Perform the following tasks in AWS and submit screenshots of each step as evidence:

(i) Launch multiple EC2 instances (web servers) and configure them in different Availability Zones.

(ii) Create an Application Load Balancer (ALB) to distribute traffic across these instances.

(iii) Configure health checks so that faulty instances are automatically removed from load balancing.

(iv) Enable Auto Scaling to add/remove instances based on traffic demand.

(v) Configure path-based routing: /auth requests go to the authentication service, /order requests go to the order processing service. Integrate the Load Balancer with Route 53 so that global users are routed to the nearest AWS region.

1️⃣ Launch EC2 Instances in Multiple Availability Zones

→ Go to **AWS Management Console → EC2 → Launch Instance**
→ Choose **Amazon Linux 2 AMI** (or preferred OS)
→ Select **Instance Type** → e.g., t2.micro (Free Tier)
→ In **Network Settings**, choose:

- **VPC** → your default or custom VPC
- **Subnet** → choose *different subnets* for each instance (each in a different Availability Zone for high availability)
  → Under **Security Group**, create or select one named web-server-sg

  Inbound security group:

ssh-> source type: my IP

http-> anywhere

- **Advanced details:**

  USER DATA CODE:

  #!/bin/bash

  yum update -y

  yum install -y httpd

  systemctl start httpd

  systemctl enable httpd

  echo "<h1>Hello from $(hostname -f)</h1>" > /var/www/html/index.html

  Launch the first instance instance. Repeat the same with a new key-pair

---

2️⃣ Create an Application Load Balancer (ALB)

→ Go to **EC2 → Load Balancers → Create Load Balancer**
→ Choose **Application Load Balancer**
→ Select **Scheme:** internet-facing
→ Select **VPC and Subnets** used by your EC2 instances
→ Create a new **Security Group** alb-sg with a description: Allows SSH access to developers

ADD INBOUND RULES:

- Allow inbound **HTTP (80)** – source: anywhere ipv4; and **HTTPS (443) same** from 0.0.0.0/0

  Then create the security group. Now go back and select it.

  → Under **Listeners**, choose:

- HTTP (port 80) → Forward to a new **Target Group. Create new target group:** target type-instances, give name my-app-target, then next. Now select both instances -> include as pending below->next->create target group. **Now go back and add it**
  → Click **Create Load Balancer**

ITLL WORK NOWW!!!!!!!!

Hello from ip-172-31-36-67.eu-north-1.compute.internal

NOW WE CAN GO TO STEP 4, LEAVE 3

3️⃣ Configure Health Checks

→ Go to **Target Groups → Create Target Group**
→ Type → **Instances**

→ Define a **Health Check Path** → e.g., /index.html
→ Set:

- Healthy threshold: 2
- Unhealthy threshold: 2
  → Register your running EC2 instances with this Target Group
  → Return to the ALB and **attach** this Target Group as the default target

---

🔼 Enable Auto Scaling

→ Go to **EC2 → Launch Templates → Create Launch Template**
→ Add:

NAME: my-lb-template

CHECK Provide guidance to help me....

- AMI (Amazon Linux 2)
- Instance Type (e.g., t3.micro); select an already created keypairs from the two
- Security Group → web-server-sg but I did launch-wizard-1
- Go to **advanced details and paste:**

- #!/bin/bash
- yum update -y
- yum install -y httpd
- systemctl start httpd
- systemctl enable httpd
- echo "<h1>Hello from $(hostname -f)</h1>" > /var/www/html/index.html

  Now create launch template-> view->**ACTIONS-> CREATE AUTO SCALING GROUP**

  → Create Auto Scaling Group my-lb-autoscalegrp
  → Choose your Launch Template my-lb-da; SELECT VERSION AS LATEST->next
  → Select the same **VPC(already chosen) and subnets** used earlier
  → Attach the ASG to your **ALB Target Group ie select Attach to an existing load balancer-> select my-app-target**

  Now turn on elastiv load balancing heal checks in health checks->next

  → Define:

- Minimum: 1
- Desired: 2
- Maximum: 4
  → **Add Target tracking Scaling Policy → Target Tracking** → choose metric like *Average CPU Utilization* = 50%->next
  → Review and **Create Auto Scaling Group**

5️⃣ Configure Path-Based Routing

→ Go to **EC2 → Load Balancers → Select your load balancer (click on mylb) → Listeners tab**
→ Choose your **HTTP:80 listener → Manage rules-> add rules**

In CONDITIONS->add conditions:path:

Path condition value: /auth* then choose target group below (in forward to target groups) -> next-> priority: 10-> next->add rule

Do again for:

Path condition value: /order* then choose target group below (in forward to target groups) -> next-> priority: 20-> next->add rule

Optional/right way-
→ Create **Two Target Groups**:

- auth-service-tg → For authentication service
- order-service-tg → For order processing service
  → Add **Listener Rules**:
- If path is /auth* → Forward to auth-service-tg
- If path is /order* → Forward to order-service-tg
  → Save the rules and test by accessing your ALB DNS with the specific paths.

6️⃣ Integrate with Route 53 for Global Routing

→ Prerequisite: Deploy your full setup (EC2, ALB, ASG) in **at least two AWS regions** (e.g., N. Virginia & Singapore).
→ Go to **Route 53 →Create Hosted Zones → Select your domain kikiweb.com -> create hosted zone**
→ Click **Create Record ->** subdomain: app
→ **Alias check on -> choose endpoint: alias to application and classic load balancer, choose region as own current and routing policy region also the same current; record id: stocklolm load balancer-> create reciords**

Now create another record exactly same except routing region is a new one and record id: hyd load balancer -> create records

NOW IP IS CHANGING AFTER REFRESHING!!!!

7. LAMBDA EXPERIMENT:

1. open s3-> create bucket, bucket name: test-bucket, leave public access blocked and versioning disabled -> create bucket

2. open a new tab of aws

3. aws lambda-> create a function. function name: mytestfunc, runtime: python 3.13

4. CHANGE DEFAULT EXECUTION ROLE: create a new role from aws policy templates, write role name: myTestFunctionRole. Policy Templates: select Amazon S3 object read-only permissions-> create function

5. Go to code-> code source -> edit the lambda_function.py and write the below code:

```
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')

def lambda_handler(event, context):
  # Uncomment for debugging:
  # print("Received event: " + json.dumps(event, indent=2))

  bucket = event['Records'][0]['s3']['bucket']['name']
  key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')

  try:
    response = s3.get_object(Bucket=bucket, Key=key)
    print("CONTENT TYPE: " + response['ContentType'])
    return {
      'statusCode': 200,
      'ContentType': response['ContentType']
    }
  except Exception as e:
    print(e)
    print(f'Error getting object {key} from bucket {bucket}. '
        'Make sure they exist and your bucket is in the same region as this function.')
    raise e
```

then click on deploy

6. + add trigger (upar) : select a source- s3; select bucket: test-bucket -> i acknowledge-> add

7. go back to test-bucket-> PROPERTIES: event notifications ss take

8. go back to lambda page and open triggers tab below in configuration - take ss

9. come back to test-bucket page and select upload option-> add files -> upload a png file-> upload

10. now go to lambda page and check monitor tab->take ss of graphs-> view cloud watch logs-> click on log streams blue link below and take ss of log streams

Application.py

```
from flask import Flask, render_template, request, redirect, url_for

application = Flask(name)
app = application

# Simple in-memory task list
tasks = []

@app.route('/')
def index():
    return render_template('index.html', tasks=tasks)

@app.route('/add', methods=['POST'])
def add_task():
    task = request.form.get('task')
    if task:
        tasks.append(task)
    return redirect(url_for('index'))

@app.route('/delete/<int:task_id>')
def delete_task(task_id):
    if 0 <= task_id < len(tasks):
        tasks.pop(task_id)
    return redirect(url_for('index'))

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

CODES:

login page:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>Login Page</title>
 <style>
   body {
    font-family: "Poppins", sans-serif;
    display: flex;
    height: 100vh;
    align-items: center;
    justify-content: center;
    background: #f4f6ff;
   }
```

```
    .login-container {
      background: white;
      padding: 2rem;
      border-radius: 1rem;
      box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
      width: 300px;
      text-align: center;
      transition: all 0.3s ease;
    }
    input {
      width: 100%;
      padding: 10px;
      margin: 10px 0;
      border: 1px solid #ddd;
      border-radius: 8px;
    }
    button {
      width: 100%;
      padding: 10px;
      border: none;
      background: #4c5bd4;
      color: white;
      border-radius: 8px;
      cursor: pointer;
    }
    button:hover {
      background: #3c4ac2;
    }
    .success {
      color: #2e7d32;
      font-weight: bold;
      font-size: 18px;
      padding: 10px;
    }
  </style>
</head>
<body>
  <div class="login-container" id="loginBox">
    <h2>Login</h2>
    <form id="loginForm">
      <input type="email" id="email" placeholder="Email" required />
      <input type="password" id="password" placeholder="Password" required />
      <button type="submit">Login</button>
    </form>
  </div>

  <script>
    const form = document.getElementById("loginForm");
```

```
  const loginBox = document.getElementById("loginBox");

  form.addEventListener("submit", (e) => {
   e.preventDefault(); // prevent page reload
   loginBox.innerHTML = <div class="success">✅ Login Successful!</div>;
   setTimeout(() => {
    loginBox.innerHTML = <h2>Welcome!</h2><p>You are now logged in.</p>;
   }, 1500);
  });
 </script>
</body>
</html>
```

application form:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>Application Form</title>
 <style>
  body {
   font-family: Poppins, sans-serif;
   background: #f9fafc;
   display: flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
  }
  form {
   background: white;
   padding: 2rem;
   border-radius: 1rem;
   width: 320px;
   box-shadow: 0 4px 10px rgba(0,0,0,0.1);
   transition: all 0.3s ease;
  }
  h2 {
   text-align: center;
   margin-bottom: 1rem;
   color: #333;
  }
  label {
   display: block;
   margin: 8px 0 4px;
   font-size: 14px;
   color: #555;
  }
```

```
  input, select {
    width: 100%;
    padding: 8px;
    border: 1px solid #ccc;
    border-radius: 8px;
  }
  button {
    width: 100%;
    margin-top: 15px;
    background: #5563de;
    color: white;
    border: none;
    padding: 10px;
    border-radius: 8px;
    cursor: pointer;
  }
  button:hover {
    background: #4652d0;
  }
  .success {
    background: white;
    padding: 2rem;
    border-radius: 1rem;
    box-shadow: 0 4px 10px rgba(0,0,0,0.1);
    text-align: center;
    font-size: 18px;
    color: #2e7d32;
    font-weight: 600;
  }
 </style>
</head>
<body>
 <div id="formContainer">
  <form id="applicationForm">
   <h2>Application Form</h2>

   <label for="name">Full Name</label>
   <input type="text" id="name" required />

   <label for="email">Email</label>
   <input type="email" id="email" required />

   <label for="role">Preferred Role</label>
   <select id="role">
    <option>Frontend Developer</option>
    <option>Backend Developer</option>
    <option>UI/UX Designer</option>
   </select>
```

```
    <label for="resume">Upload Resume</label>
    <input type="file" id="resume" />

    <button type="submit">Submit</button>
   </form>
  </div>

  <script>
   const form = document.getElementById("applicationForm");
   const container = document.getElementById("formContainer");

   form.addEventListener("submit", (e) => {
    e.preventDefault(); // Prevent form reload
    container.innerHTML = `
     <div class="success">
      ✅ Application Submitted Successfully!<br><br>
      Thank you for applying 💼
     </div>
    `;
   });
  </script>
</body>
</html>

profile section:
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Profile Page</title>
 <style>
  body {
   font-family: "Poppins", sans-serif;
   background: #f5f7ff;
   color: #333;
   display: flex;
   justify-content: center;
   align-items: center;
   min-height: 100vh;
   margin: 0;
  }
  .profile-container {
   background: white;
   padding: 2rem 2.5rem;
   border-radius: 1.5rem;
   box-shadow: 0 6px 16px rgba(0,0,0,0.1);
```

```
  width: 400px;
}
.profile-header {
  text-align: center;
  margin-bottom: 1.5rem;
}
.profile-header img {
  width: 100px;
  height: 100px;
  border-radius: 50%;
  object-fit: cover;
  margin-bottom: 10px;
}
.profile-header h2 {
  margin: 0;
  font-size: 22px;
  color: #2c2f7b;
}
.profile-header p {
  margin: 5px 0;
  color: #777;
  font-size: 14px;
}
.section {
  margin-top: 1.5rem;
}
.section h3 {
  border-bottom: 2px solid #e0e2f8;
  padding-bottom: 5px;
  color: #4b54c5;
  font-size: 18px;
}
ul {
  list-style-type: none;
  padding-left: 0;
}
li {
  margin: 8px 0;
  font-size: 15px;
}
.skills span {
  display: inline-block;
  background: #eef0ff;
  color: #4b54c5;
  padding: 5px 10px;
  border-radius: 8px;
  margin: 4px;
  font-size: 13px;
```

```
  }
 </style>
</head>
<body>
 <div class="profile-container">
  <div class="profile-header">
   <img src="https://i.pravatar.cc/100" alt="Profile Picture">
   <h2>Abhi Rawat</h2>
   <p>Frontend Developer | UI/UX Designer</p>
   <p>Email: abhi@example.com</p>
  </div>

  <div class="section bio">
   <h3>About Me</h3>
   <p>I’m a passionate frontend developer with a keen eye for design and detail. I love creating clean, responsive, and user-friendly interfaces using modern web technologies like React and JavaScript.</p>
  </div>

  <div class="section skills">
   <h3>Skills</h3>
   <span>HTML</span>
   <span>CSS</span>
   <span>JavaScript</span>
   <span>React.js</span>
   <span>UI/UX Design</span>
  </div>

  <div class="section projects">
   <h3>Projects</h3>
   <ul>
    <li><strong>E-commerce:</strong> A stylish frontend store built with React and CSS for seamless shopping.</li>
    <li><strong>Music Platform:</strong> Interactive music web app focusing on UI accessibility and engagement.</li>
    <li><strong>Hotel Management System:</strong> A Flutter + MySQL system with full software lifecycle implementation.</li>
   </ul>
  </div>
 </div>
</body>
</html>
```