Employee Sync: Cloud-Based Employee Management System with AWS, CustomTkinter, and RDS

Below is a comprehensive guide to creating an Employee Management System in Python using CustomTkinter for the GUI, Amazon RDS (Relational Database Service) for the database, and AWS resources for hosting. We'll then deploy it to your GoDaddy domain (`learn.xyz`) using AWS infrastructure. This guide assumes you have basic familiarity with Python, AWS, and web hosting.

Overview of the Process

1. 'Set Up AWS Environment': Configure AWS RDS for the database and an EC2 instance for hosting the backend.

2. 'Develop the Employee Management System': Write Python code with CustomTkinter for the GUI and connect it to RDS.

3. 'Deploy the Application': Use AWS EC2 to host the backend and connect it to your GoDaddy domain

4. 'Configure Domain': Point your GoDaddy domain to the AWS-hosted application.

Step 1: Set Up AWS Environment

#1.1 Create an AWS Account

- Go to aws.amazon.com and sign up if you don’t already have an account.
- Log in to the AWS Management Console.

#1.2 Set Up Amazon RDS

1. Navigate to the 'RDS' service in the AWS Console.
2. Click 'Create Database':
   - Choose 'Standard Create'.
   - Select 'MySQL' as the engine.
   - Choose the 'Free Tier' template (for testing).
   - Settings:
     - DB instance identifier: `employee-db`
     - Master username: `admin`
     - Master password: Set a secure password (e.g., `MySecurePass123`).
   - Leave other settings as default for now (e.g., DB instance size: `db.t2.micro`).

3. After creation, note the 'Endpoint' (e.g., `employee-db.xxxxx.us-east-1.rds.amazonaws.com`) and 'Port' (default: 3306).

4. Create a database and table:

- Use a MySQL client (e.g., MySQL Workbench) to connect to the RDS instance.

- Run this SQL to set up the database:

```
[ CREATE DATABASE employees;
 USE employees;
 CREATE TABLE empdata (
     id INT PRIMARY KEY,
     name VARCHAR(255),
     age INT,
     role VARCHAR(255)
 );
 ]
```

#1.3 Set Up an EC2 Instance

1. Navigate to the 'EC2' service in the AWS Console.
2. Click 'Launch Instance':
   - Choose 'Amazon Linux 2 AMI' (free tier eligible).
   - Instance type: `t2.micro` (free tier).
   - Create a new key pair (e.g., `employee-key.pem`) and download it.
   - Configure security group:
     - Allow SSH (port 22) from your IP.
     - Allow HTTP (port 80) from anywhere (`0.0.0.0/0`).
3. Launch the instance and note its 'Public IPv4 address' (e.g., `54.123.45.67`).

Step 2: Develop the Employee Management System

#Prerequisites

- Install Python 3.x on your local machine.
- Install required libraries:

```bash
pip install customtkinter mysql-connector-python
```

#Source Code

Here’s the Python code for the Employee Management System using CustomTkinter and connecting to AWS RDS:

```python
import customtkinter as ctk
import mysql.connector
from mysql.connector import Error

# Database configuration for AWS RDS
db_config = {
    'host': 'employee-db.xxxxx.us-east-1.rds.amazonaws.com',  # Replace with your RDS endpoint
    'user': 'admin',
    'password': 'MySecurePass123',  # Replace with your RDS password
    'database': 'employees'
}

# Function to connect to RDS
def connect_db():
    try:
        connection = mysql.connector.connect('db_config)
        if connection.is_connected():
            return connection
    except Error as e:
        print(f"Error connecting to RDS: {e}")
        return None

# Function to add an employee
def add_employee():
    connection = connect_db()
    if connection:
        cursor = connection.cursor()
        try:
            id_val = id_entry.get()
            name_val = name_entry.get()
            age_val = age_entry.get()
            role_val = role_entry.get()
            sql = "INSERT INTO empdata (id, name, age, role) VALUES (%s, %s, %s, %s)"
            cursor.execute(sql, (id_val, name_val, age_val, role_val))
            connection.commit()
            result_label.configure(text="Employee added successfully!")
        except Error as e:
            result_label.configure(text=f"Error: {e}")
        finally:
            cursor.close()
            connection.close()

# Function to view all employees
def view_employees():
    connection = connect_db()
    if connection:
        cursor = connection.cursor()
        try:
            cursor.execute("SELECT * FROM empdata")
            rows = cursor.fetchall()
            result_text = "\n".join([f"ID: {row[0]}, Name: {row[1]}, Age: {row[2]}, Role: {row[3]}" for row in rows])
            result_label.configure(text=result_text if rows else "No employees found.")
        except Error as e:
            result_label.configure(text=f"Error: {e}")
        finally:
            cursor.close()
            connection.close()

# Set up the GUI
app = ctk.CTk()
app.title("EmployeeSync - Employee Management System")
app.geometry("600x500")

# Fonts
font1 = ("Arial", 20, "bold")
```

```
font2 = ("Arial", 15)

# Input fields
ctk.CTkLabel(app, text="ID:", font=font1).place(x=20, y=20)
id_entry = ctk.CTkEntry(app, font=font2, width=200)
id_entry.place(x=100, y=20)

ctk.CTkLabel(app, text="Name:", font=font1).place(x=20, y=80)
name_entry = ctk.CTkEntry(app, font=font2, width=200)
name_entry.place(x=100, y=80)

ctk.CTkLabel(app, text="Age:", font=font1).place(x=20, y=140)
age_entry = ctk.CTkEntry(app, font=font2, width=200)
age_entry.place(x=100, y=140)

ctk.CTkLabel(app, text="Role:", font=font1).place(x=20, y=200)
role_entry = ctk.CTkEntry(app, font=font2, width=200)
role_entry.place(x=100, y=200)

# Buttons
ctk.CTkButton(app, text="Add Employee", command=add_employee, font=font1).place(x=20, y=260)
ctk.CTkButton(app, text="View Employees", command=view_employees, font=font1).place(x=20, y=320)

# Result label
result_label = ctk.CTkLabel(app, text="", font=font2, wraplength=500)
result_label.place(x=20, y=380)

# Start the app
app.mainloop()
```

#Notes

- Replace the `db_config` values with your actual RDS endpoint and credentials.
- This is a desktop GUI app. To make it web-accessible, we’ll deploy a Flask backend in the next step.

Step 3: Deploy the Application on AWS EC2

#3.1 Connect to EC2

1. Use SSH to connect to your EC2 instance:

   ```bash
   ssh -i employee-key.pem ec2-user@54.123.45.67  # Replace with your public IP
   ```

#3.2 Install Dependencies

2. Update the instance and install Python, pip, and other tools:

   ```bash
   sudo yum update -y
   sudo yum install python3 -y
   sudo yum install python3-pip -y
   pip3 install customtkinter mysql-connector-python flask
   ```

#3.3 Create a Flask Backend

Since CustomTkinter is for desktop GUIs, we'll create a simple Flask web app to serve the employee management system online. Save this as `app.py` on your EC2 instance:

```python
from flask import Flask, request, render_template
import mysql.connector
from mysql.connector import Error

app = Flask(__name__)

# RDS configuration
db_config = {
    'host': 'employee-db.xxxxx.us-east-1.rds.amazonaws.com',  # Replace with your RDS endpoint
    'user': 'admin',
    'password': 'MySecurePass123',  # Replace with your RDS password
    'database': 'employees'
```

```
}


def connect_db():
    try:
        connection = mysql.connector.connect('db_config)
        if connection.is_connected():
            return connection
    except Error as e:
        print(f"Error: {e}")
        return None


@app.route('/', methods=['GET', 'POST'])
def index():
    message = ""
    if request.method == 'POST':
        id_val = request.form['id']
        name_val = request.form['name']
        age_val = request.form['age']
        role_val = request.form['role']
        connection = connect_db()
        if connection:
            cursor = connection.cursor()
            try:
                sql = "INSERT INTO empdata (id, name, age, role) VALUES (%s, %s, %s, %s)"
                cursor.execute(sql, (id_val, name_val, age_val, role_val))
                connection.commit()
                message = "Employee added successfully!"
            except Error as e:
                message = f"Error: {e}"
            finally:
                cursor.close()
                connection.close()

    # Fetch all employees
    connection = connect_db()
    employees = []
    if connection:
        cursor = connection.cursor()
        cursor.execute("SELECT * FROM empdata")
        employees = cursor.fetchall()
        cursor.close()
        connection.close()

    return render_template('index.html', employees=employees, message=message)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

#3.4 Create an HTML Template

Create a folder named `templates` in the same directory as `app.py`, and add this `index.html`:

```html
<!DOCTYPE html>
<html>
<head>
    <title>EmployeeSync</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        table { border-collapse: collapse; width: 100%; }
        th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
        th { background-color: #f2f2f2; }
    </style>
</head>
<body>
    <h1>EmployeeSync - Employee Management System</h1>
    <form method="POST">
        <label>ID: <input type="text" name="id" required></label><br><br>
        <label>Name: <input type="text" name="name" required></label><br><br>
        <label>Age: <input type="text" name="age" required></label><br><br>
        <label>Role: <input type="text" name="role" required></label><br><br>
        <input type="submit" value="Add Employee">
    </form>
    <p>{{ message }}</p>
    <h2>Employee List</h2>
    <table>
        <tr><th>ID</th><th>Name</th><th>Age</th><th>Role</th></tr>
        {% for emp in employees %}
        <tr><td>{{ emp[0] }}</td><td>{{ emp[1] }}</td><td>{{ emp[2] }}</td><td>{{ emp[3] }}</td></tr>
        {% endfor %}
    </table>
</body>
</html>
```

#3.5 Upload Files to EC2

1. Use SCP to upload `app.py` and the `templates` folder:

   ```bash
   scp -i employee-key.pem app.py ec2-user@54.123.45.67:/home/ec2-user/
   scp -i employee-key.pem -r templates ec2-user@54.123.45.67:/home/ec2-user/
   ```

#3.6 Run the Flask App

2. On the EC2 instance, run the app:

   ```bash
   python3 app.py
   ```

3. Test it by visiting `http://54.123.45.67` in your browser.

#3.7 Keep the App Running

Use `nohup` to run the app in the background:

```bash
nohup python3 app.py &
```

<u>**Step 4: Configure GoDaddy Domain**</u>

1. 'Log in to GoDaddy':

- Go to your GoDaddy account and manage your domain

2. 'Update DNS Settings':

- Go to the DNS management page.
- Set the 'A Record' to point to your EC2 instance's public IP (e.g., `54.123.45.67`):
  - Host: `@`
  - Points to: `54.123.45.67`
  - TTL: 1 hour (default).
- Save the changes.

3. 'Wait for DNS Propagation':

- It may take 24-48 hours for the DNS changes to propagate.

4. 'Test the Domain':

- Once propagated, visit `http://learn.xyz` to see your Employee Management System live.

Additional Notes

- 'Security': For production, secure your app with HTTPS using an SSL certificate (e.g., via AWS Certificate Manager and Elastic Load Balancer).
- 'Scalability': Consider using AWS App Runner or Elastic Beanstalk for easier deployment and scaling.
- 'Backup': Regularly back up your RDS database using AWS snapshots.

AUTHOR: - NIHAR PADHI

BATCH: - INHYD57