Step-by-Step Guide: Deploying a Full Stack Website to Grasp Three-Tier Architecture on AWS Cloud

For this project, I created a full-stack app utilizing Python (Flask) 🐍 for the backend, MySQL 🗄️ for the database, and Docker 🐬 to containerize the application. On the frontend, I worked with HTML 📝, CSS 🎨, and JavaScript (JSON for data handling) 🛠️ to create a simple yet functional user interface.

Why is this app a good candidate for 3-tier architecture? The 3-tier architecture allows for a clear separation of concerns:

1-Presentation Layer: The frontend (HTML, CSS, JavaScript) handles the user interface, making it easier to manage and update the user experience independently of the backend.

2-Application Layer: The backend (Flask) processes user inputs, handles business logic, and serves the necessary data from the database.

3-Data Layer: The database (MySQL) securely stores and retrieves data, ensuring data integrity and accessibility.

This separation makes the application scalable, maintainable, and more secure—ideal characteristics for deploying on cloud platforms like AWS ☁️.

📚 To make the process even easier, I've provided a step-by-step guide that walks through how to deploy the app locally or on AWS. This guide should help anyone interested in exploring 3-tier architecture, whether you're deploying on a local machine or in the cloud.

Caution

⚠️ **Important Warning:** **Before deploying this on AWS, make sure you're using a free tier account. AWS services can incur charges quickly, sometimes unexpectedly, if you're not careful. I learned this the hard way ☹️, and it ended up costing me. Even if you're reading the step-by-step details provided, I highly recommend educating yourself on AWS pricing and cost management strategies. Just taking the time to read up on these topics can save you from potential surprises and help you avoid unnecessary expenses.**

🔗 You can download the app from my GitHub link https://github.com/Sara951/3-tier-architecture-web and run it if you'd like. Although the application is simple, the main purpose of this guide is to provide a hands-on learning experience with AWS 3-Tier Architecture, making it an effective way to master the concepts.

1. Understanding the 3-Tier Architecture

A 3-tier architecture typically consists of three layers:
- Presentation Layer (Web Tier): This is where the user interface resides. It includes the front-end components of your application that interact with users. In AWS, this is usually hosted on EC2 instances or AWS Elastic Beanstalk.
- Application Layer (Logic Tier): This is the middle layer where the application logic is processed. It often involves back-end servers or application servers that process data from the presentation layer and pass it to the database layer.
- Data Layer (Database Tier): This is the back-end layer where the database resides. It stores and manages data for the application.

2. Analyzing the Application Structure

The application directory contains the following key components:
Backend Directory:
- app.py: This is likely the main application file, probably a Python-based server (possibly using Flask or Django).
- Dockerfile: This file contains instructions to build a Docker image for the backend.
- init_db.sql: This SQL file is probably used to initialize the database schema or data.
- requirements.txt: This file lists the Python dependencies required by the backend application.

Frontend Directory:

- app.js: This likely contains the JavaScript logic for the front-end of the application.
- Dockerfile: This file contains instructions to build a Docker image for the frontend.
- index.html: The main HTML file for the web application.
- style.css: The CSS file for styling the front-end.

3. Detailed Deployment Steps

1. Setting Up the Data Layer (Database) with RDS

1. Create an RDS Instance:
- Navigate to the RDS console in the AWS Management Console.
- Click 'Create Database.'
- Select 'MySQL' as the engine type (or any other supported database).
- Choose the 'Free Tier' template if you're testing, or select the instance class that fits your needs.
- Set the database identifier, master username, and password.
- In the 'Connectivity' section, select the VPC and ensure it can communicate with your application layer.
- Configure the Security Group to allow inbound connections only from your backend EC2 instance.
- Review and launch the instance.

2. Initialize the Database:
- Once the RDS instance is up, connect to it using an SQL client or from your backend EC2 instance.
- Run the init_db.sql script to set up your database schema.
```bash
mysql -h <RDS-endpoint> -u <username> -p < init_db.sql
```

2. Setting Up the Application Layer (Backend) on EC2

1. Create an EC2 Instance:
- Navigate to the EC2 Dashboard and launch a new EC2 instance.
- Select an appropriate Amazon Machine Image (AMI), like Amazon Linux 2 or Ubuntu.
- Choose an instance type (e.g., t2.micro).
- Configure instance details, including the VPC that can communicate with the RDS instance.
- Configure the security group to allow inbound traffic on the port used by your application (e.g., 5000 for Flask).
- Review and launch the instance.

2. Set Up the Backend Application:
- SSH into the instance.

- Install Docker if it's not already installed:
```bash
sudo yum update -y
sudo yum install docker -y
sudo service docker start
sudo usermod -a -G docker ec2-user
```
- Clone your application repository or transfer your code.
- Navigate to the backend directory and build the Docker image:
```bash
cd backend
docker build -t backend-app .
```
- Run the Docker container:
```bash
docker run -d -p 5000:5000 backend-app
```
- Ensure the backend can connect to the RDS database. Update the app.py file or environment variables to point to the RDS instance.

3. Setting Up the Presentation Layer (Frontend) on EC2

1. Create an EC2 Instance:
- Follow similar steps as the backend instance but for the frontend.
- Ensure the security group allows HTTP traffic on port 80.

2. Set Up the Frontend Application:
- SSH into the instance.
- Install Docker:
```bash
sudo yum update -y
sudo yum install docker -y
sudo service docker start
sudo usermod -a -G docker ec2-user
```
- Navigate to the frontend directory and build the Docker image:
```bash
cd frontend
docker build -t frontend-app .
```
- Run the Docker container:
```bash
```

```
docker run -d -p 80:80 frontend-app
```

4. Connecting the Tiers

- Application to Database: Ensure that the backend application connects to the RDS instance. This may involve setting the appropriate environment variables or configuring connection strings in the app.py file.
- Frontend to Application: Ensure the frontend communicates with the backend by configuring the correct API endpoints in the app.js or related files.

5. Testing and Final Adjustments

- Access your frontend using the public IP or DNS of your frontend EC2 instance.
- Ensure it interacts with the backend, and the backend correctly queries the database.
- Adjust security groups, load balancers, or auto-scaling groups as necessary based on your application's performance and security needs.