Deployment of 3-Tier Applications Using AWS Services

A Traditional Cloud Deployment for a Web Application with Database Backend

Domain: Cloud Computing and DevOps

Project Overview:
This project demonstrates a manually deployed 3-tier architecture on AWS for a web application where users can submit data, which is then stored in a backend database. The setup does not use DevOps tools (like Jenkins, Docker, or Kubernetes) and relies on AWS native services for hosting, networking, and storage.

GitHub URL: https://github.com/staragile2016/AWS-3-Tier-Architecture.git

Key Components:

- Presentation Tier (Frontend) – Hosted on AWS S3 (Static Website).
- Application Tier (Backend) – Runs on AWS EC2 (Web Server).
- Database Tier – Managed by AWS RDS (MySQL/PostgreSQL).
- Networking &amp; Security – VPC, IAM, Route 53 for access control and DNS.

Business Requirements:
Users should be able to submit data via a web form.
Data should be stored securely in a backend database.
The application should be accessible via a custom domain (using Route 53).
Security best practices should be followed (IAM roles, VPC isolation).

1. Presentation Tier (Frontend - S3):

- A static website (HTML, CSS, JavaScript) hosted on AWS S3.
- Users interact with a web form that sends data to the EC2 backend.
- S3 is configured for static web hosting with public read access.

2. Application Tier (Backend - EC2):

- An EC2 instance running a web server (Apache/Nginx) with a backend application (Node.js/Python/Java).
- The application processes user requests and stores data in RDS.
- Security Groups restrict traffic only from S3 (frontend) and RDS (database).

3. Database Tier (RDS):

- A managed RDS instance (MySQL/PostgreSQL) for storing user data.
- Deployed in a private subnet for security.
- Accessible only by the EC2 application server.

4. Networking &amp; Security (VPC, IAM, Route 53):
VPC Setup:

- Public Subnet (EC2, S3)
- Private Subnet (RDS)

IAM Roles:

- EC2 instance has permissions to access RDS.
- S3 has restricted access policies.

Route 53:

- Custom domain name

Step-by-Step Manual Deployment:
1. Set Up VPC &amp; Subnets:

- Create a VPC with:
- Public Subnet (for EC2 &amp; S3)
- Private Subnet (for RDS)
- Configure Internet Gateway for public access.
- Set up NAT Gateway (if needed for private subnet outbound traffic).

2. Deploy S3 Static Website (Frontend):

- Create an S3 bucket with static website hosting enabled.
- Upload HTML, CSS, and JavaScript files.
- Set bucket policy for public read access.

3. Launch EC2 Instance (Backend):

- Choose an Amazon Linux/Ubuntu AMI.
- Install web server (Apache/Nginx) and backend runtime (Node.js/Python).
- Configure Security Group to allow HTTP/HTTPS traffic from S3.
- Assign an IAM role with RDS access permissions.

4. Set Up RDS Database:

- Launch an RDS instance (MySQL/PostgreSQL) in the private subnet.
- Configure Security Group to allow only EC2 connections.
- Note down endpoint, username, and password.

5. Connect Frontend to Backend:

- Modify the frontend (S3 JavaScript) to send API requests to the EC2 backend.
- Example: Javascript
  fetch (&quot;http://&lt;EC2-Public-IP&gt;/submit-data&quot;, { method:
  &quot;POST&quot;, body: JSON.stringify(data) });

6. Configure Route 53 (Custom Domain):

- Register a domain (or use an existing one).
- Create an A record pointing to the S3 static website endpoint.

- (Optional) Set up HTTPS using ACM (AWS Certificate Manager).

7. Test the Application:

- Users access the website via the Route 53 domain.
- Data entered in the form is sent to EC2, which stores it in RDS.
- Verify data persistence in the RDS database.

Expected Outcome:

✅ Fully Functional 3-Tier Web App – Users can submit data, which is stored in RDS.
✅ Secure &amp; Isolated Architecture – VPC, private subnets, and IAM roles ensure security.
✅ Custom Domain Access – Route 53 provides a user-friendly URL.
✅ No DevOps Tools Used – Entirely manual AWS setup.