AWS 3-Tire-Project

🚀 \*\*Building a Scalable and Secure 3-Tier Architecture on AWS\*\* 🚀

The purpose of this project was to design and deploy a scalable and secure 3-tier architecture on AWS in which you can also store data in structure format. By leveraging AWS services such as EC2, S3, RDS, VPC, and ELB, the goal was to create a robust infrastructure capable of hosting a web application with high availability, fault tolerance, and efficient resource management. This setup facilitates seamless traffic management, secure data access, and optimized performance across the web, application, and database layers. Here's a breakdown of the steps I took:

\*\*Step 1: VPC and Subnet Configuration\*\*

- Created a VPC with a CIDR block of 10.0.0.0/16.

- Set up six subnets distributed across two availability zones for web, app, and database layers.

\*\*Step 2: Internet Gateway and NAT Gateways\*\*

- Attached is an Internet Gateway (IGW) to the VPC.

- Created two NAT Gateways to ensure outbound internet traffic for private subnets.

\*\*Step 3: Routing Configuration\*\*

- Configured public and private route tables to manage traffic flow between subnets, IGW, and NAT Gateways.

\*\*Step 4: Security Groups\*\*

- Defined security groups for web, internal load balancer, app, and database layers with specific inbound and outbound rules to ensure secure communication between layers.

\*\*Step 5: S3 Bucket for Code Storage\*\*

- Created an S3 bucket to store application code and configurations.

\*\*Step 6: Database Configuration\*\*

- Set up a subnet group and deployed an Aurora database instance within private subnets.

\*\*Step 7: Application Tier Setup\*\*

- Launched EC2 instances and connected them to the database.

- Installed necessary dependencies and configured the application using Node.js and PM2 for process management.

- Downloaded application code from S3 and started the app, ensuring database connectivity and functionality.

\*\*Step 8: Internal Load Balancer Configuration\*\*

- Created an Application Load Balancer (ALB) with internal scheme.

- Configured target groups and health checks to ensure high availability and load balancing.

\*\*Step 9: Web Tier Setup\*\*

- Launched EC2 instances for the web layer and installed necessary components.

- Downloaded web tier code from S3 and built the React application.

- Configured NGINX as a web server and proxy to the internal load balancer.

\*\*Outcome\*\*

- Successfully implemented a scalable, secure, and highly available 3-tier architecture on AWS.

- Ensured seamless integration and communication between the web, app, and database layers.

- Achieved high availability through load balancing and efficient routing.

This project was a great learning experience and a testament to the power of AWS services in building modern, resilient infrastructure.

**#AWS** **#CloudComputing** **#DevOps** **#3TierArchitecture** **#VPC** **#EC2** **#S3** **#RDS** **#Nginx** **#LoadBalancing** **#InfrastructureAsCode**