Deployment Architecture Documentation

1. Overview

This deployment architecture document describes the deployment setup for a cloud-based social networking application. The frontend is built with React and communicates with a Flask-based backend API. Authentication is handled via Keycloak, and all services are hosted on AWS infrastructure.

2. Components

- React Frontend: Client-side UI application.
- Flask Backend: RESTful API developed using Flask and SQLAlchemy ORM.
- Keycloak: External identity provider for handling authentication and token issuance.
- PostgreSQL (or MySQL): Relational database managed via Amazon RDS.
- AWS ECS: Container orchestration to deploy Flask app.
- AWS ELB: Load balancing and routing traffic to Flask ECS service.
- AWS VPC: Isolated virtual network for deploying services.

3. Data Flow

1. Users interact with the React frontend.
2. The frontend communicates with the Flask backend through HTTP requests routed via AWS ELB.
3. Flask validates tokens with Keycloak before serving data.
4. Validated requests interact with the RDS database for storing/retrieving posts, comments, etc.

4. Security

- HTTPS via SSL termination on ELB.
- JWT tokens managed by Keycloak.
- IAM roles control access to AWS resources.
- VPC subnets and security groups isolate backend services.

5. Scalability

- ELB enables horizontal scaling.
- ECS can autoscale container instances.
- RDS supports read replicas for scaling reads.

Deployment Architecture Documentation

6. Fault Tolerance

- Multi-AZ RDS deployment for high availability.

- ELB distributes traffic across healthy ECS tasks.

- React app can be served from S3 + CloudFront for resilience.

7. Monitoring & Logging

- AWS CloudWatch for metrics and logs.

- Keycloak admin dashboard for auth activity.

- ECS service logs captured via CloudWatch Logs.

8. Future Enhancements

- Add Redis for caching.

- Use S3 for storing images/media.

- Integrate CI/CD pipeline using AWS CodePipeline.