Week 1 Progress Report: Secure Cloud Application Architecture

Aastik Bansal [23114002] Abdullah Azeem [23114003]

February 20, 2026

1 Executive Summary

We have successfully completed the objectives for Week 1 of the "Secure Cloud Application Architecture" project. The core infrastructure has been deployed using Docker, establishing a secure environment with segmented networks. Key security features including SSL/TLS termination, JWT-based authentication, and basic Role-Based Access Control (RBAC) foundation have been implemented.

2 Architecture Implementation

The system follows the proposed defense-in-depth architecture with the following components currently running:

2.1 Infrastructure & Networking

- **Containerization**: All services (Client, Gateway, App Server, Database) are containerized using Docker.
- **Network Segmentation**:
  - `public_net`: Connects Client → API Gateway.
  - `dmz_net`: Connects API Gateway → Application Server (Internal Access Only).
  - `internal_net`: Connects Application Server → Database (Isolated from Gateway/Public).
- **Service Management**: Created `start_services.sh` and `stop_services.sh` for reliable lifecycle management.

2.2 Component Status

Component	Status & Details
Client	**Operational**. Simple HTML/JS interface for Login/Register. Served via Nginx.
API Gateway	**Operational**. Nginx acting as Reverse Proxy. Handles **SSL/TLS** (Self-Signed) and Rate Limiting.
App Server	**Operational**. FastAPI backend. Implements Business Logic and Auth.
Database	**Operational**. PostgreSQL 15. Stores Users and Roles. Persistent storage via volumes.

Table 1: Component Status Summary

3 Security Features Implemented

3.1 Authentication & Authorization

- **JWT (JSON Web Tokens)**:
  - Users can Register and Login.
  - Successful login returns an `access_token` containing the user's `role`.
  - Passwords are hashed using **Bcrypt** before storage.
- **RBAC Foundation**:
  - Database schema supports `role` field (default: 'user').
  - Tokens carry role information for future endpoint protection.

3.2 Network Security

- **SSL/TLS Termination**:
  - API Gateway listens on port 443 (mapped to 8443).
  - Encrypts traffic between Client and Gateway.
- **Reverse Proxying**:
  - Direct access to the App Server and Database is blocked from the host (except via mapped ports for debugging, which are restricted in production).
  - All API requests are routed through `/api` on the Gateway.

4 Challenges & Resolutions

- **Self-Signed Certificates**: Browsers flag the certificate as insecure.
  Resolution: Added instructions to bypass the warning for development testing.
- **Service Orchestration**: Ensuring Database is ready before App Server starts.
  Resolution: Implemented a retry mechanism in `main.py` (`get_engine` with retries).

5 Next Steps (Week 2)

The foundation is stable. We are ready to proceed to **Task 3: Automated Threat Modeling**:

1. Verify exact Trust Boundaries.
2. Simulate attacks (Brute Force, SQLi).
3. Enhance Logging for the Security Module.