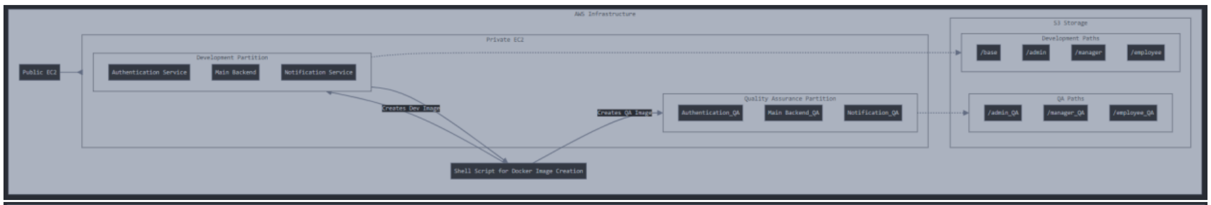
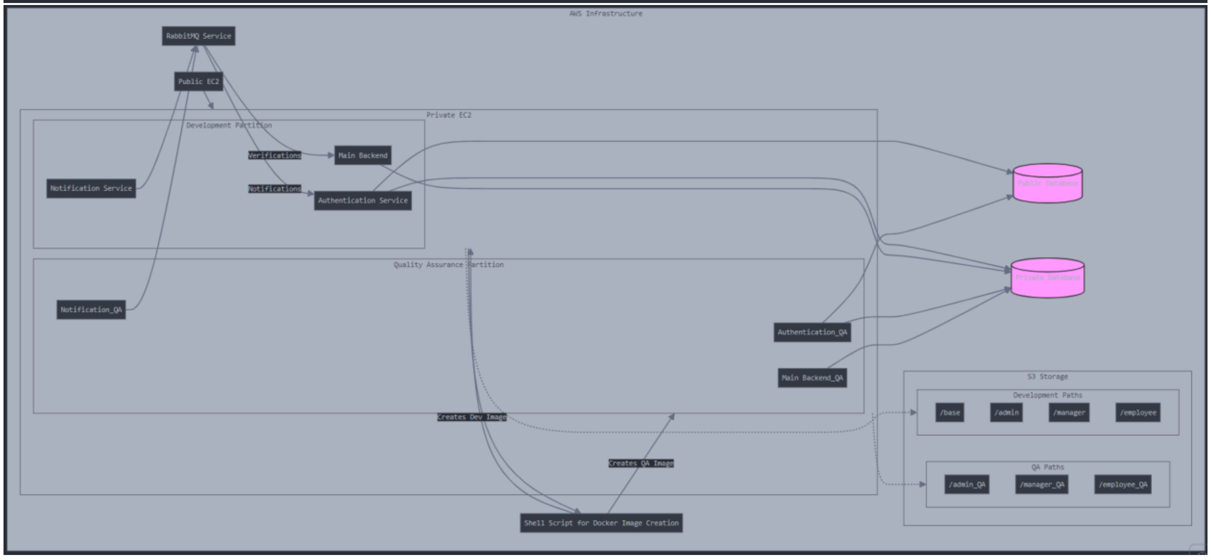
Complete flow of Deployment





1. AWS EC2 (Elastic Compute Cloud) Setup

- **Private EC2**: Private EC2 instances are often placed in a **private subnet** for better security, as they are not directly accessible from the internet.
- **Public EC2**: It might be used for internal processes or handling sensitive data. This is why it's partitioned into two environments:
    - **Development Partition**: Contains components ("Authentication," "Main Backend," "Notification") where developers work on features or updates.
    - **Quality Assurance (QA) Partition**: Has the same components, but in QA versions ("Authentication_QA," "Main Backend_QA," "Notification_QA"), where the QA team tests features for stability before release.

2. Docker Usage

- Instead of using a CI/CD pipeline, we rely on **shell scripts** to automate Docker image creation.
    - When code is updated in any of the Development partition services, two Docker images are created:
        - One image for the **Development team**.
        - One image for the **QA team**.

- This process likely ensures that both teams work with the most recent code changes in their respective environments so QA team start testing and development team go on for further development.

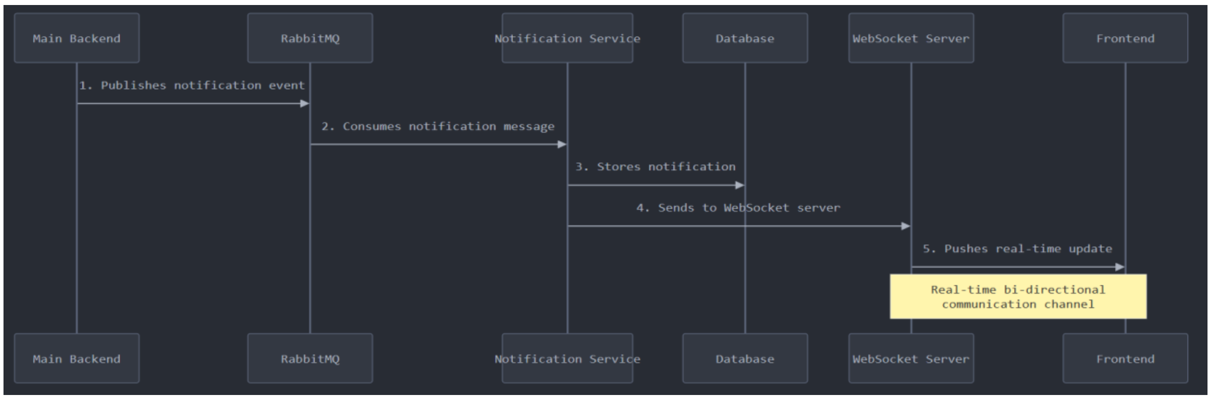
3. S3 (Simple Storage Service)

- S3 is being used for code storage in both Development and Quality Assurance environments.
    - Paths in **Development**: /base, /admin, /manager, /employee.
    - Paths in **QA**: /admin_QA, /manager_QA, /employee_QA.
- S3 ensures that data is separated and properly organized for each environment.

4. Database and RabbitMQ

- Both the **databases** and **RabbitMQ** services are hosted **outside** EC2 and S3.
    - **RabbitMQ** seems to be used for handling notifications and verifications (likely processing tasks or events in the background).
    - The **databases** are probably shared resources across both environments (Development and QA), ensuring that both teams have access to the same data when required.

Notification path


Main Backend
RabbitMQ
Notification Service
Database
WebSocket Server
Frontend
1. Publishes notification event
2. Consumes notification message
3. Stores notification
4. Sends to WebSocket server
5. Pushes real-time update
Real-time bi-directional communication channel

- **Main Backend (MBE) → RabbitMQ**:
    - Backend generates a notification event
    - Could be for things like new messages, alerts, updates, etc.
    - Publishes to specific RabbitMQ exchange/queue

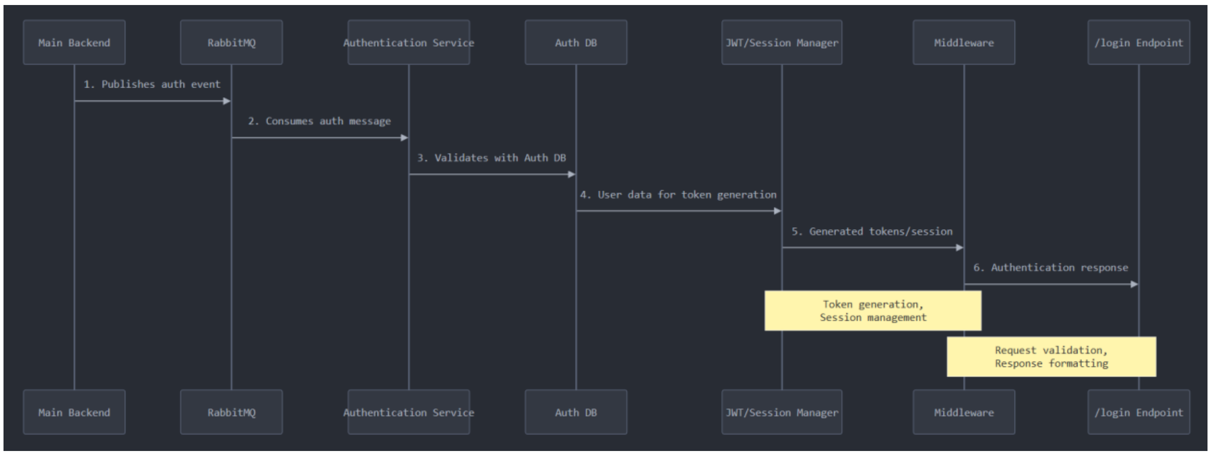
- **RabbitMQ → Notification Service**:
    - Message queuing ensures reliable delivery
    - Handles high volume of notifications
    - Provides message persistence

- **Notification Service → Database**:
    - Stores notification for historical record
    - Enables notification retrieval if user is offline
    - Helps with notification state management

- **Notification Service → WebSocket (WS)**:
    - Maintains persistent connections with clients
    - Enables real-time bi-directional communication
    - More efficient than polling for real-time updates

- **WebSocket → Frontend (FT)**:
    - Receives real-time updates via WebSocket connection
    - Updates UI immediately without page refresh
    - Could show notifications as toasts, alerts, or in a notification centre

Authentication path


Main Backend
RabbitMQ
Authentication Service
Auth DB
JWT/Session Manager
Middleware
/login Endpoint
1. Publishes auth event
2. Consumes auth message
3. Validates with Auth DB
4. User data for token generation
5. Generated tokens/session
6. Authentication response
Token generation, Session management
Request validation, Response formatting

- **Main Backend → RabbitMQ:**

    - Initiates authentication request
    - Could be login attempt, token refresh, etc.

- **RabbitMQ → Authentication Service:**

    - Queues authentication messages
    - Ensures reliable delivery
    - Handles high volume of auth requests

- **Authentication Service → Auth DB:**

    - Validates user credentials
    - Retrieves user data/permissions

- **Auth DB → JWT/Session Manager:**

    - Handles:
        - Token generation (JWT)
        - Session creation
        - Token signing
        - Permission attachment

- **JWT/Session Manager → Middleware:**

  - Responsibilities:
    - Request validation
    - Header processing
    - CORS handling
    - Rate limiting
    - Security checks

- **Middleware → /login Endpoint:**

  - Final response formatting
  - Includes:
    - Access token
    - Refresh token (if applicable)
    - User data
    - Permissions
    - Session information