Cloud-Based Manhwa Web Application

Project Overview

This project uses AWS cloud services to deploy a web application that displays a list of fantasy manhwa titles, their genres, and descriptions. The key components of the project include EC2 instances for hosting the Node.js application, S3 for storing static assets (images), PM2 for managing the Node.js process, and AWS Auto Scaling with a Load Balancer to ensure high availability and scalability.

1. EC2 Instance Usage

The project runs a Node.js backend on an **EC2 (Elastic Compute Cloud)** instance, which is the core hosting environment for the web application. The EC2 instance is part of AWS's cloud infrastructure and provides scalable computing capacity, allowing the application to serve users efficiently.

- **Instance Type**: A lightweight instance type (e.g., **t2.micro**) is selected to keep initial costs low, but the infrastructure is designed to scale up based on demand.
- **Operating System**: Amazon Linux 2 or Ubuntu is used, providing a stable environment for running the Node.js application.
- **Process Management**: **PM2** is employed to keep the Node.js app running continuously, manage crashes, and restart the app when necessary. This guarantees high uptime and ensures the application stays available even if issues occur.
- **Security Groups**: The EC2 instance is secured by allowing only HTTP, HTTPS, and restricted SSH access.

2. S3 Bucket Usage

An **S3 (Simple Storage Service)** bucket stores static assets, such as manhwa cover images. These assets are accessed via public URLs and are integrated into the web application's frontend to display relevant images for each manhwa title.

- **Public Access**: Images stored in the S3 bucket are made publicly accessible to allow the frontend of the application to retrieve and display them.
- **Scalability**: S3 provides virtually unlimited storage, making it an ideal solution for hosting and serving static files, ensuring the application can handle an increasing number of images without performance degradation.
- **Cost Efficiency**: S3 is cost-effective for static asset storage, with pricing based on the amount of data stored and retrieved.

3. Load Balancing

This project uses an **Application Load Balancer (ALB)** to distribute incoming traffic across multiple EC2 instances. The load balancer improves fault tolerance and ensures the application remains available even when individual EC2 instances are under heavy load or become unavailable.

- **Traffic Distribution**: The ALB routes user requests to the EC2 instances that are healthy and capable of handling the traffic. This helps avoid overloading any single instance and ensures a smoother user experience.
- **Health Checks**: The ALB performs regular health checks on all registered EC2 instances to verify they are running properly. If an instance fails a health check, it is temporarily removed from the pool of available instances until it becomes healthy again.
- **HTTPS Support**: The load balancer supports **HTTPS** by integrating an SSL certificate from AWS Certificate Manager (ACM). This ensures secure communication between users and the web application by encrypting all traffic.

By spreading the traffic load evenly and dynamically adjusting the available resources, the load balancer ensures the application scales efficiently during periods of high demand.

4. Auto Scaling Overview

Auto Scaling is a critical component of this project's cloud infrastructure. It ensures the application can handle fluctuating traffic patterns by dynamically adjusting the number of EC2 instances based on real-time demand.

- **Scaling Policies**: The project uses CPU utilization as the primary metric for scaling. When CPU usage exceeds a predefined threshold (e.g., 70%), Auto Scaling automatically launches additional EC2 instances to distribute the workload. Conversely, when CPU usage drops below a certain threshold (e.g., 30%), unnecessary instances are terminated to reduce costs.
- **Minimum and Maximum Instances**: The Auto Scaling group is configured with a **minimum** and **maximum** number of instances to prevent over-provisioning or under-provisioning. For example:
  - **Minimum**: The application always runs with at least one EC2 instance to ensure availability.
  - **Maximum**: The application can scale up to five instances to handle peak traffic loads, but no more to control costs.
- **Elasticity**: Auto Scaling ensures the infrastructure is highly elastic. During traffic spikes, the application can quickly add instances to maintain performance. When traffic subsides, Auto Scaling scales down the infrastructure to conserve resources and minimize costs.
- **High Availability**: By automatically adjusting resources based on traffic, Auto Scaling ensures the web application maintains high availability. Users will always have a responsive experience, even if the demand suddenly increases.

5. Security Measures

Security is enforced through **Security Groups** for both the EC2 instances and the Load Balancer. These security groups define which ports and protocols can be accessed and restrict unnecessary exposure.

- **Inbound Traffic**: The application only allows inbound traffic on ports **80 (HTTP)** and **443 (HTTPS)**. SSH access (port 22) is restricted to the administrator's IP address for security reasons.
- **HTTPS**: The application uses **HTTPS** to secure all communication between the client and the server, ensuring that sensitive data is transmitted securely.