A CLOUD COMPUTING CASE STUDY

Olivia Jin

Introduction

MovieHive

Project OverView:

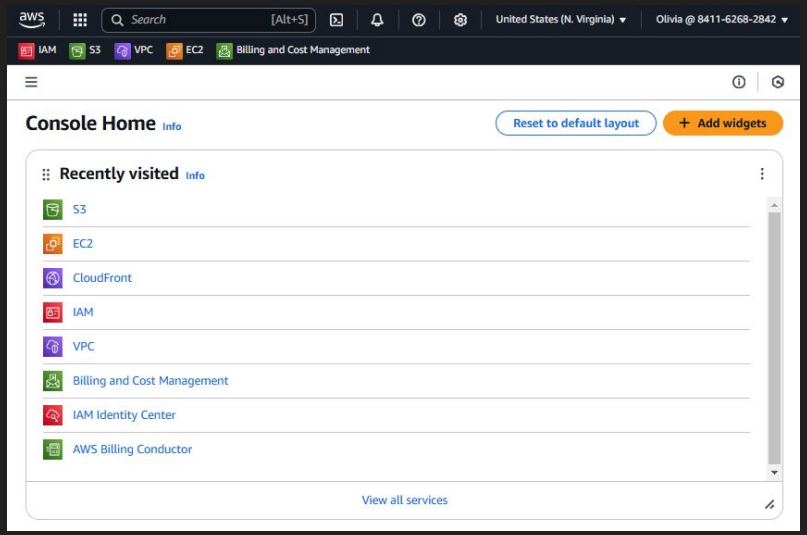
- MovieHive is a Web Application that provides movie details and recommendation based on AWS.

Main Objectives:

- Allow users to search for movies and add them to their favorites.

PURPOSE

Building a scalable cloud-based movie website. Deploying on AWS instead of a triational local environment to enhance scalability and maintainability. Aiming to provide faster load times and a more stable service compared to traditional websites.



FEATURES

Scalable Server Infrastructure

Utilizing AWS EC2 instances to separate the backend API server and database server

Static File Hosting & Performance

Hosting static resources (images, CSS, JavaScript) using AWS S3 to optimize loading speed

Security & API Management

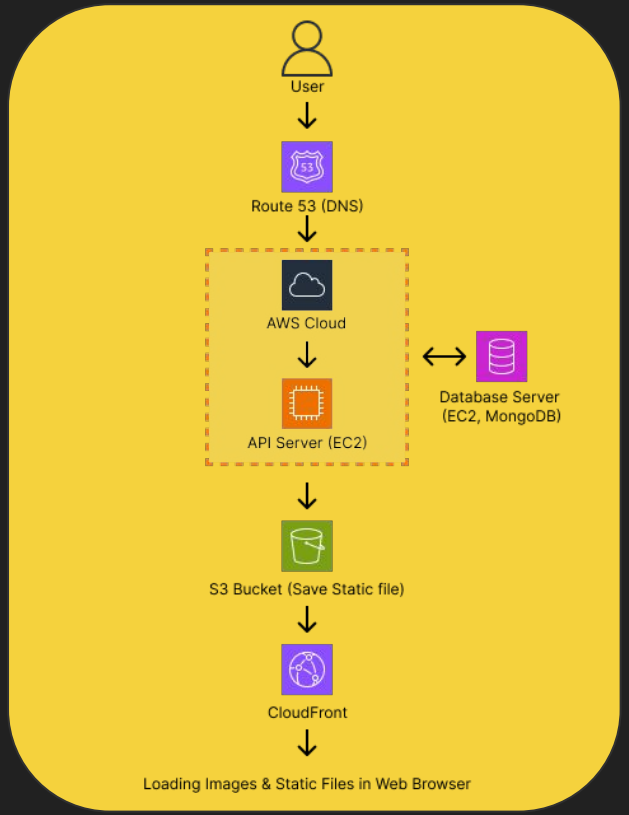
Protecting the API server through AWS Security Groups and IAM role management

CLOUD ARCHITECTURE

Users access the website through their browser, and AWS Route 53 (DNS service) directs their requests to the appropriate server. The EC2 API server processes these requests and interacts with other services to retrieve data.

The database stores movie details, user information, and favorite lists, while AWS S3 holds static content like movie posters. The API server or frontend fetches these images and delivers them to users.

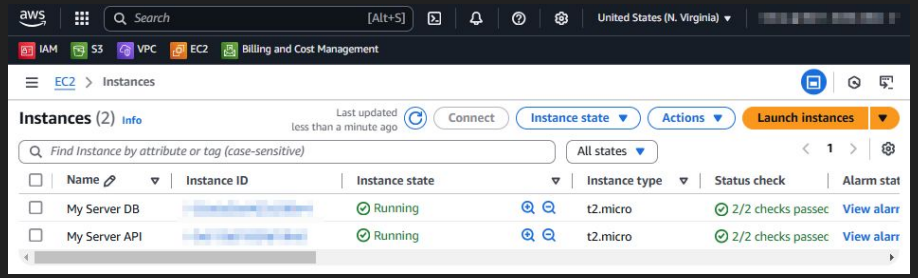
To ensure fast and reliable content delivery, CloudFront (CDN) distributes static and dynamic content across different regions, optimizing the loading speed of images and other assets in the web browser.

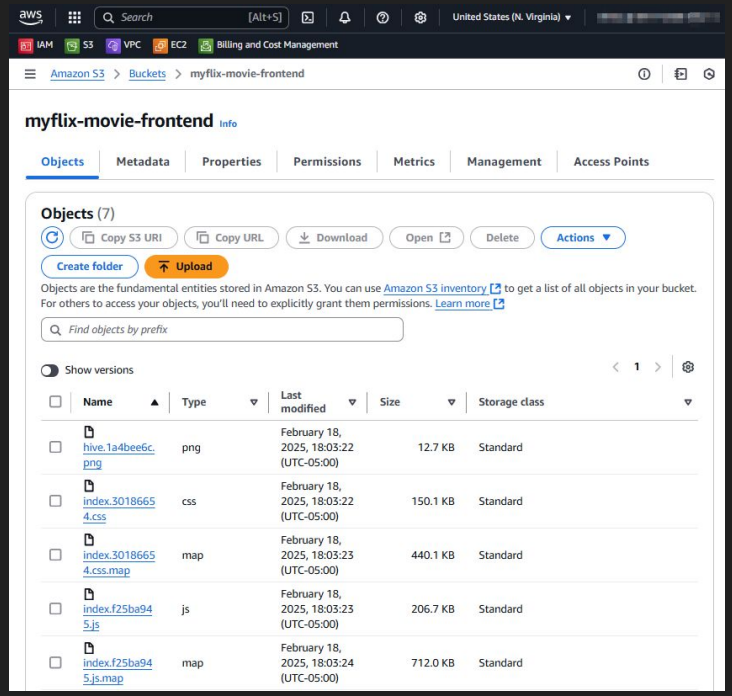


TECHNOLOGY STACK

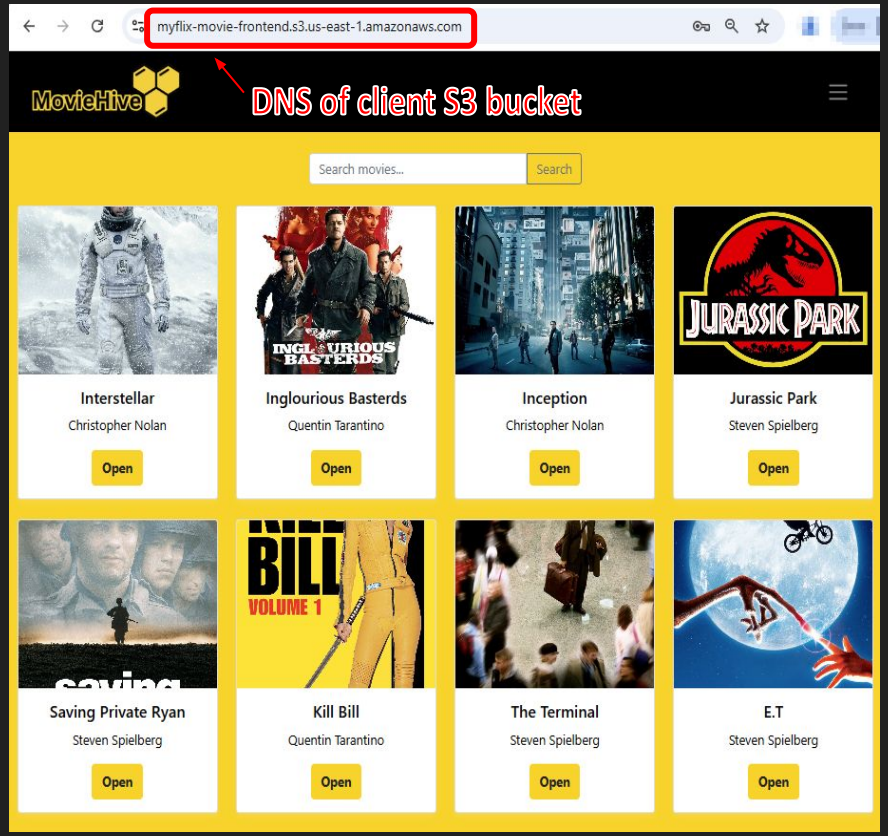
- Frontend: React, SCSS
- Backend: Node.js, MongoDB
- Cloud Service: AWS (EC2, S3), Heroku
- Database: MongoDB Atlas
- CD/CI: GitHub

Setup EC2 Instances (API and DB Instances)

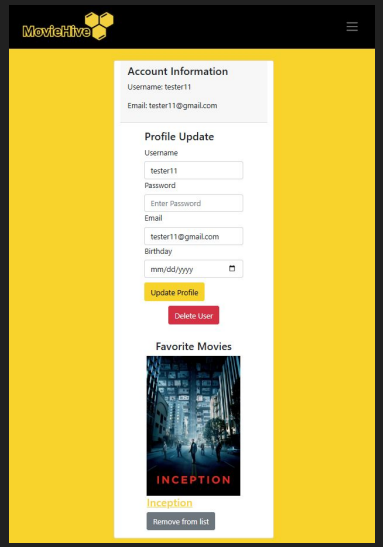




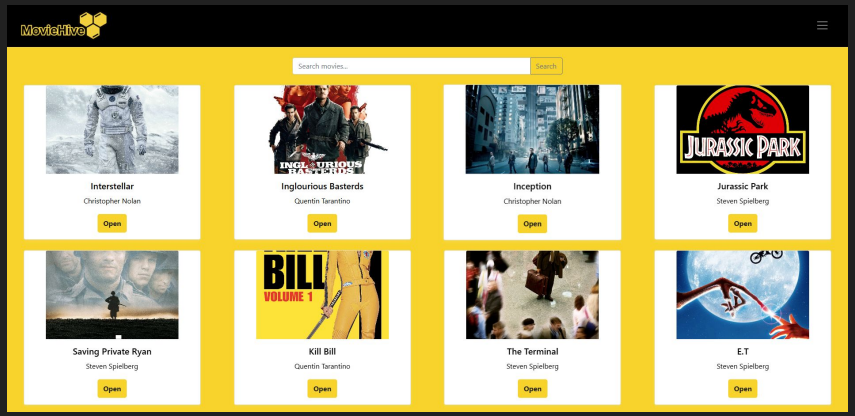
AWS Infrastructure Setup



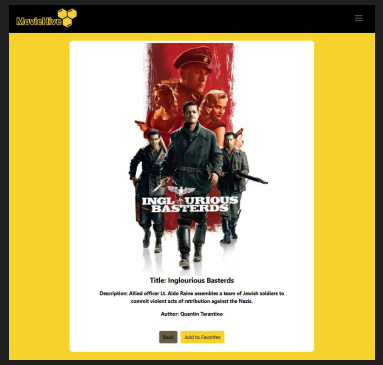
PAGE VIEWS



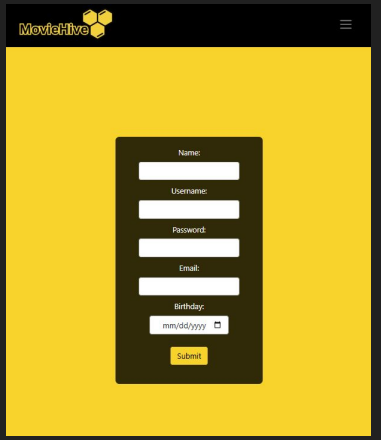
Profile View



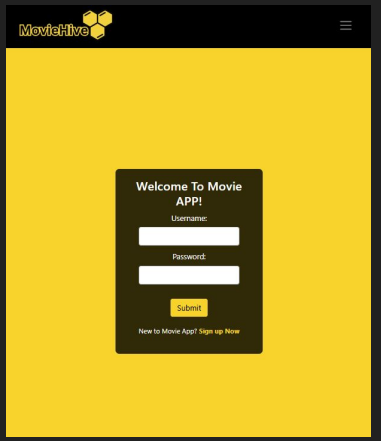
Main View



Movie View



Sign-up View



Login View