CSCI 5409
CLOUD COMPUTING

TERM PROJECT REPORT

Banner ID: B00981502

Table of Contents

What I have built and what it is supposed to do.

In the term project, I built a web application using React and AWS services that will allow me to stream movies over the internet. This project is called “Movies Ocean”. To use this application, the first user will have to create an account handled using AWS Cognito. Once the user has been registered, they can watch all the movies on the website. The videos and images are stored in the S3 bucket. Then there is an admin panel also where admins can upload new videos to the website. This application supports role-based functionality. The information about all new uses and their roles are stored in the AWS RDS using APIs created by API gateway and lambda. All the URLs of media content will be read and updated to the database present in the AWS RDS using APIs. Also, this website supports authorization validation on the APIs so that the appropriate users are the only ones who will be able to access the APIs.

How you met your menu item requirements.

Compute

Services I used

- AWS EC2
- AWS Lambda

Reason for choosing this over other options.

- Here I have used EC2 for deploying my application on the internet. The reason for choosing EC2 is that it provides more control over resources compared to AWS Elastic Beanstalk. It also provides more flexibility which opens room for improved optimization.
- AWS Lambda is perfect for event-driven workloads, where I need to run code in response to HTTP requests, making it easier to maintain and scale without server management overhead. Its seamless integration with other AWS services and pay-per-use pricing model makes it a cost-effective solution for handling backend logic. Also, this is a fairly simple application and its alternative such as AWS Step Function is suitable for more complex applications. This is the reason why I selected this over others.

Storage

Services I used

- AWS S3
- AWS Relational Database Service (RDS)

Reason for choosing this over other options.

- AWS S3 is widely famous for its scalability and cost-effectiveness. When it comes to storing large media files, S3 is the best solution available. Also, it offers high availability and durability which makes it a perfect choice. Compared to Elastic File Storage, it has the property that it can be used with different cloud providers.
- AWS RDS is a fully managed service, and it provides performance with automatic failover, read replicas, and multi-AZ deployments for high availability and reliability. The main reason for choosing this over AWS Aurora is the cost-effectiveness. It is very affordable compared to AWS Aurora.

Network

Services I used

- AWS API Gateway

Reason for choosing this over other options.

- Amazon API Gateway provides a fully managed service for creating, deploying, and managing APIs at scale. It integrates seamlessly with AWS Lambda, ensuring secure and efficient routing of API requests. Application Load Balancer is better suited for distributing traffic to multiple targets (e.g., EC2 instances) rather than API management. Also, it is highly scalable.

General

Service I used

- AWS Cognito
- AWS Backup

Reason for choosing this over other options.

1. AWS Cognito

Ease of Integration:

- **Seamless Integration with AWS Services**: Integrates effortlessly with other AWS services like API Gateway and Lambda.
- **SDKs and Libraries**: Provides SDKs for multiple platforms including JavaScript, making it easy to integrate with your React frontend.

User Management:

- **User Pools and Identity Pools**: Supports user pools for authentication (sign-up and sign-in) and identity pools for authorization (access control).
- **Customizable Authentication Flows**: Allows customization of user registration and login workflows, including multi-factor authentication (MFA) and password policies.

Security:

- **Built-in Security Features**: Provides features like secure password storage, automatic handling of token refresh, and user account recovery.
- **Compliance**: Meets various compliance requirements, which is crucial for handling sensitive user information.

Scalability:

- **Automatic Scaling**: Scales automatically handle a large number of users without additional configuration or infrastructure management.
- **High Availability**: Offers high availability and reliability, ensuring that authentication services are always accessible.

Cost-Effective:

- **Pay-as-You-Go Pricing**: Offers a pricing model based on the number of users and operations, which can be more cost-effective than developing and maintaining a custom authentication system.

2. AWS Backup

Centralized Backup Management:

- **Unified Console**: Provides a single console to manage backups across multiple AWS services, simplifying backup management.
- **Consistent Policies**: Allows the application of consistent backup policies across resources, ensuring comprehensive data protection.

Automation and Scheduling:

- **Automated Backups**: Supports automated backup scheduling, reducing the need for manual intervention and minimizing the risk of missed backups.
- **Retention Policies**: Enables setting retention policies to automatically delete old backups, helping to manage storage costs and compliance requirements.

Reliability and Durability:

- **Data Integrity**: Ensures data integrity and availability with multiple copies stored in different availability zones.

- **Compliance and Security**: Complies with various industry standards and regulations, providing encrypted backups to secure your data.

Cost-Effective:

- **Optimized Storage Costs**: Offers cost-effective storage options for backups, with pay-as-you-go pricing based on the storage consumed.
- **Eliminates Need for Third-Party Solutions**: Reduces the need for third-party backup solutions, which can be more expensive and complex to integrate.

Restore and Disaster Recovery:

- **Ease of Restore**: Provides straightforward restore procedures, minimizing downtime and ensuring business continuity in case of data loss.
- **Point-in-Time Recovery**: Supports point-in-time recovery, enabling precise restoration of data to a specific state.

A description of the deployment model you chose for your system

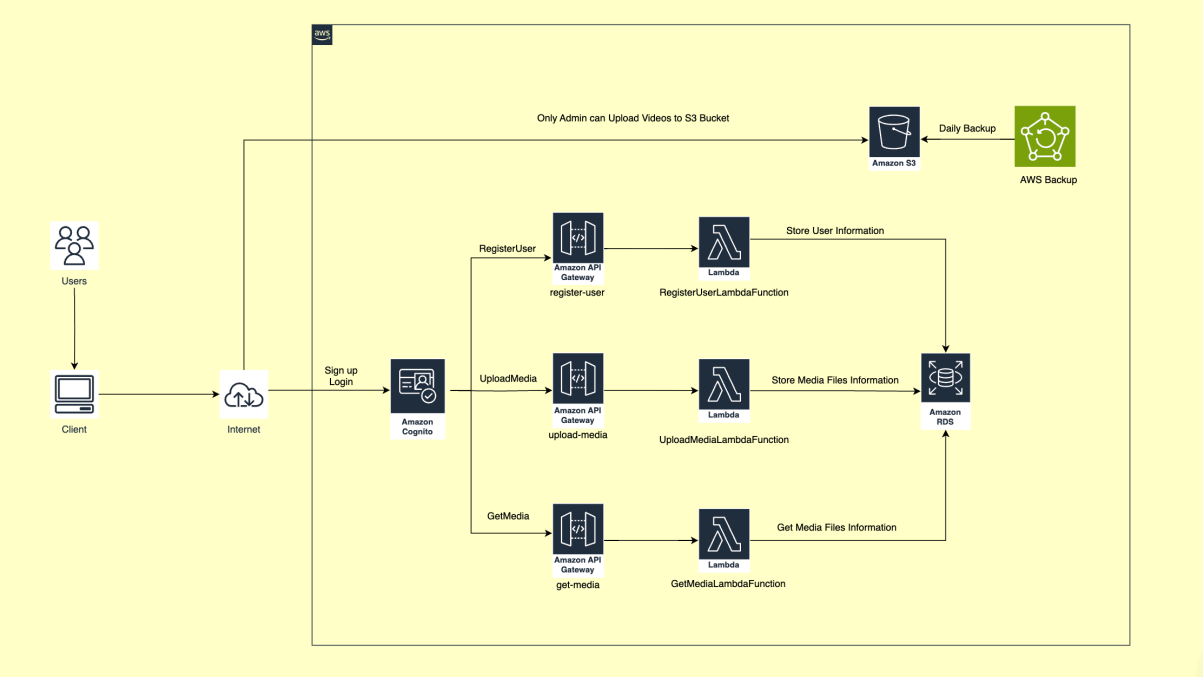
This system falls under the public cloud deployment model. This is because this is accessible by everyone using the internet. There are no restrictions on accessing the system which is why it is a public cloud deployment model.

A description of the delivery model you chose for your system

This system is a Software as a Service(SaaS). Here is why I think this is the SaaS model.

This system is hosted on AWS, making it accessible via the internet without users needing to install software locally. Users can access the service through a web browser, which is typical for React-based applications. Most online streaming services operate on a subscription basis, which aligns with the SaaS model. AWS provides the infrastructure to scale your service based on demand, a common feature of SaaS applications.

The architecture of the final system.



An analysis of your project's approach to security

Data in Transit:

1. HTTPS/TLS: HTTPS is used to make API calls to the API gateway. This enhances the security and it passes encrypted data over the internet.
2. API Gateway: The use of Amazon API Gateway ensures secure API endpoints.

Data at Rest:

1. Amazon S3:
    - Server-side encryption is enabled while creating the s3 bucket.
    - Here AWS manages the keys for the encryption and makes sure that the data is stored are encrypted.

2. Lambda Functions:
    - I have used environment variables for sensitive information which enhances the security.

Additional Security Measures:

1. Authentication:
    - Amazon Cognito is used for user authentication, which is a secure, managed service.

2. Authorization:
    - Authorization is implemented to make sure that the API calls are made by a legitimate user using Cognito pool Authorizer.

An analysis of the cost metrics for operating your system.

Up-Front Costs

1. **Domain Name Registration**: If you need a custom domain, the cost typically ranges from $10 to $50 per year.
2. **SSL Certificate**: To secure your website, an SSL certificate can cost between $0 (Let's Encrypt) to $100+ per year for premium certificates.
3. **Initial Development and Setup**: Assuming you are developing the system yourself, the main costs here are your time and any software/tools you might need.

Ongoing Costs

1. **Amazon S3**:
    - **Storage Cost**: $0.023 per GB per month for the first 50 TB.
    - **Data Transfer**: $0.09 per GB for data transferred out beyond the free tier limit.
2. **Amazon RDS**:
    - **Instance Cost**: Varies based on the instance type. For a small DB instance, the cost can be around $0.017 per hour (approximately $12.24 per month).
    - **Storage Cost**: $0.10 per GB per month for provisioned storage.
3. **AWS Lambda**:
    - **Invocation Cost**: $0.20 per 1 million requests.
    - **Duration Cost**: $0.00001667 for every GB-second used.
4. **Amazon API Gateway**:
    - **Requests Cost**: $3.50 per million requests.
    - **Data Transfer**: Charges based on the amount of data transferred out.
5. **Amazon CloudFront**:
    - **Data Transfer**: $0.085 per GB for the first 10 TB per month.
    - **Requests Cost**: $0.0075 per 10,000 HTTP/HTTPS requests.
6. **AWS Backup**:
    - **Backup Storage Cost**: $0.05 per GB per month for backup storage.
7. **Amazon Cognito**:
    - **Active Users Cost**: $0.0055 per MAU (monthly active user) beyond the free tier of 50,000 MAUs.

Additional Costs

1. **Data Transfer Out**: Additional charges may apply if you have significant data transfer out of AWS.
2. **Enhanced Monitoring**: Optional costs for detailed monitoring and logging services like AWS CloudWatch.
3. **Support Plans**: If you choose an AWS Support plan, the cost can vary based on the level of support needed.

Cost Estimation Example

Let's estimate the monthly costs based on some assumptions:

- **Storage in S3**: 500 GB
- **Data Transfer Out**: 100 GB
- **RDS Instance**: t3.micro
- **Lambda Invocations**: 100000 requests
- **API Gateway Requests**: 100000 requests
- **CloudFront Data Transfer**: 500 GB
- **Cognito Users**: 1000 MAUs

Amazon S3:

- Storage: 500 GB * $0.023 = $11.50
- Data Transfer: 100 GB * $0.09 = $9.00

Amazon RDS:

- Instance Cost: $12.24
- Storage: 20 GB * $0.10 = $2.00

AWS Lambda:

- Requests: 5 * $0.20 = $1.00
- Duration (assuming 128 MB and 100 ms per invocation): $0.00001667 * 100000 * (128/1024) * 0.1 = $0.02

Amazon API Gateway:

- Requests: 5 * $3.50 = $17.50

Amazon CloudFront:

- Data Transfer: 500 GB * $0.085 = $42.50
- Requests: 50,000 requests * $0.0075 = $3.75

AWS Backup:

- Backup Storage: 500 GB * $0.05 = $25.00

Amazon Cognito:

- Active Users: (1000 - 50,000 free) * $0.0055 = $0.00 (within free tier)

Total Monthly Cost Estimate: $11.50 (S3 Storage) + $9.00 (S3 Data Transfer) + $12.24 (RDS Instance) + $2.00 (RDS Storage) + $1.00 (Lambda Requests) + $1.04 (Lambda Duration) + $17.50 (API Gateway) + $42.50 (CloudFront Data Transfer) + $3.75 (CloudFront Requests) + $25.00 (AWS Backup) + $0.00 (Cognito Users) = **$125.53**

Alternative Approaches

1. **Using Reserved Instances for RDS**:
    - **Cost Savings**: Reserved instances can save up to 50% compared to on-demand instances.
    - **Justification**: If you have predictable database usage, reserving instances for 1 or 3 years can significantly reduce costs.
2. **Optimizing Lambda Functions**:
    - **Cost Savings**: By optimizing the memory allocation and reducing the execution time of Lambda functions, you can lower the cost.
    - **Justification**: Efficiently written functions reduce both duration and memory costs.
3. **Exploring AWS Free Tier**:
    - **Cost Savings**: Utilize AWS Free Tier where possible, especially for development and testing phases.
    - **Justification**: The AWS Free Tier offers a significant amount of free usage across various services, helping to reduce initial costs.
4. **Updating Backup Policy and using Caching:**
    - **Using Redis cache**: Using cached data to satisfy API calls will optimise the system and reduce the number of API calls.