Cloud computing assignment

For

Ghackk technology

Assignment submitted by:

SAI KUMAR.V
saikumarvankudoth@gmail.com

Cloud Computing Assignment Documentation

Project Overview

This assignment demonstrates the deployment of a web application displaying the **50 Best Telugu Movies**. The application is hosted on a cloud platform, with images stored on cloud storage services. This project involves setting up a **hosting environment** on **AWS EC2** or **Azure App Service** and storing images in **AWS S3** or **Azure Blob Storage**. The primary objective is to build a simple, scalable, and secure web application hosted on **AWS**. You will leverage AWS services like **EC2** (for hosting), **S3** (for storage, if needed), and **AWS Certificate Manager (ACM)** for enabling **HTTPS**. The app should handle mock data stored in a JSON file, presenting movie titles, genres, and brief descriptions.

1. Web Application Features

The web application displays:

- A list of **Telugu movie titles**.
- Each movie's **genre** and a brief **description**.
- Images (optional) loaded from **cloud storage**.
- Web Application developed in **HTML** language.

Web Application Code:

HTML(index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Top 30 Telugu Movies</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        /* General Styles */
        body {
            font-family: Arial, sans-serif;
            background-color: #f9f9f9;
            margin: 0;
            padding: 0;
        }

        header {
```

```
        background-color: #ff5722;
        color: white;
        padding: 10px 0;
        text-align: center;
    }

    h1 {
        margin: 0;
    }

    main {
        padding: 20px;
    }

    .movie-list {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
        gap: 15px;
    }

    .movie {
        background-color: white;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    }

    .movie h2 {
        margin-top: 0;
        font-size: 1.5em;
        color: #333;
    }

    .movie p {
        margin: 5px 0 0;
        color: #555;
    }

    footer {
        text-align: center;
        padding: 10px;
        background-color: #ff5722;
        color: white;
        position: fixed;
        bottom: 0;
```

```
            width: 100%;
        }
    </style>
</head>
<body>
    <header>
        <h1>Top 30 Telugu Movies</h1>
    </header>

    <main>
        <section class="movie-list">
            <!-- Movie List -->
            <div class="movie">
                <h2>1. Baahubali: The Beginning (2015)</h2>
                <p>Directed by: S.S. Rajamouli</p>
            </div>
            <div class="movie">
                <h2>2. Baahubali: The Conclusion (2017)</h2>
                <p>Directed by: S.S. Rajamouli</p>
            </div>
            <div class="movie">
                <h2>3. RRR (2022)</h2>
                <p>Directed by: S.S. Rajamouli</p>
            </div>
            <div class="movie">
                <h2>4. Magadheera (2009)</h2>
                <p>Directed by: S.S. Rajamouli</p>
            </div>
            <div class="movie">
                <h2>5. Arjun Reddy (2017)</h2>
                <p>Directed by: Sandeep Reddy Vanga</p>
            </div>
            <div class="movie">
                <h2>6. Ala Vaikunthapurramuloo (2020)</h2>
                <p>Directed by: Trivikram Srinivas</p>
            </div>
            <div class="movie">
                <h2>7. Mahanati (2018)</h2>
                <p>Directed by: Nag Ashwin</p>
            </div>
            <div class="movie">
                <h2>8. Eega (2012)</h2>
                <p>Directed by: S.S. Rajamouli</p>
            </div>
            <div class="movie">
```

```
        <h2>9. Jersey (2019)</h2>
        <p>Directed by: Gowtam Tinnanuri</p>
    </div>
    <div class="movie">
        <h2>10. Rangasthalam (2018)</h2>
        <p>Directed by: Sukumar</p>
    </div>
    <div class="movie">
        <h2>11. Srimanthudu (2015)</h2>
        <p>Directed by: Koratala Siva</p>
    </div>
    <div class="movie">
        <h2>12. Athadu (2005)</h2>
        <p>Directed by: Trivikram Srinivas</p>
    </div>
    <div class="movie">
        <h2>13. Pushpa: The Rise (2021)</h2>
        <p>Directed by: Sukumar</p>
    </div>
    <div class="movie">
        <h2>14. Khaleja (2010)</h2>
        <p>Directed by: Trivikram Srinivas</p>
    </div>
    <div class="movie">
        <h2>15. Nuvvu Naaku Nachav (2001)</h2>
        <p>Directed by: K. Vijaya Bhaskar</p>
    </div>
    <div class="movie">
        <h2>16. Geetha Govindam (2018)</h2>
        <p>Directed by: Parasuram</p>
    </div>
    <div class="movie">
        <h2>17. Fidaa (2017)</h2>
        <p>Directed by: Sekhar Kammula</p>
    </div>
    <div class="movie">
        <h2>18. Aarya (2004)</h2>
        <p>Directed by: Sukumar</p>
    </div>
    <div class="movie">
        <h2>19. Pelli Choopulu (2016)</h2>
        <p>Directed by: Tharun Bhascker Dhaassyam</p>
    </div>
    <div class="movie">
        <h2>20. Bommarillu (2006)</h2>
```

```
            <p>Directed by: Bhaskar</p>
        </div>
        <div class="movie">
            <h2>21. Gabbar Singh (2012)</h2>
            <p>Directed by: Harish Shankar</p>
        </div>
        <div class="movie">
            <h2>22. Manam (2014)</h2>
            <p>Directed by: Vikram Kumar</p>
        </div>
        <div class="movie">
            <h2>23. Pokiri (2006)</h2>
            <p>Directed by: Puri Jagannadh</p>
        </div>
        <div class="movie">
            <h2>24. Temper (2015)</h2>
            <p>Directed by: Puri Jagannadh</p>
        </div>
        <div class="movie">
            <h2>25. Vedam (2010)</h2>
            <p>Directed by: Krish</p>
        </div>
        <div class="movie">
            <h2>26. Kshanam (2016)</h2>
            <p>Directed by: Ravikanth Perepu</p>
        </div>
        <div class="movie">
            <h2>27. Sye (2004)</h2>
            <p>Directed by: S.S. Rajamouli</p>
        </div>
        <div class="movie">
            <h2>28. Oopiri (2016)</h2>
            <p>Directed by: Vamsi Paidipally</p>
        </div>
        <div class="movie">
            <h2>29. KGF: Chapter 1 (2018)</h2>
            <p>Directed by: Prashanth Neel</p>
        </div>
        <div class="movie">
            <h2>30. Ye Maaya Chesave (2010)</h2>
            <p>Directed by: Gautham Vasudev Menon</p>
        </div>
    </section>
</main>
```

```
    <footer>
        <p>&copy; 2024 Top Telugu Movies</p>
    </footer>
</body>
</html>
```

---

2. Hosting the Application

AWS EC2 Hosting Setup

1. **Launch EC2 Instance**:
   - Access the **AWS Management Console** and launch a new **EC2 instance**.
   - Select **Amazon Linux 2** or **Ubuntu** as the **AMI**.
   - Choose an **instance type** (e.g., **t2.micro** for free tier eligibility).
   - Configure the **security group** to allow **HTTP (port 80)** and **SSH (port 22)** traffic.
   - **Launch** the instance.
2. **Connect to EC2 Instance**:
   - Use SSH to connect to the instance from your local machine:

     ```
     bash
     Copy code
     ssh -i /path/to/your-key.pem ec2-user@your-ec2-public-ip
     ```

3. **Install Web Server**:
   - Install **Apache**:

     For **Apache**:

     ```
     bash
     Copy code
     sudo yum update -y
     sudo yum install httpd -y
     sudo systemctl start httpd
     sudo systemctl enable httpd
     ```

4. **Deploy the Application**:
   - Upload the **HTML** and **JSON** files to the web server:
     - For **Apache**, place them in `/var/www/html/`.
5. **Access the Application**:
   - Open a browser and go to `http://your-ec2-public-ip/` (public ip past in new tab) to view the deployed web application.

3. Image Storage Setup

AWS S3 Storage Setup

1. **Create an S3 Bucket**:
    - Go to **S3** in the **AWS Management Console**.
    - Click **Create Bucket**, set a unique bucket name, and select a **region**.
    - Set **bucket permissions** to allow public access to objects (if you want the images to be publicly accessible).
2. **Upload Images**:
    - Go to the bucket and click **Upload**.
    - Add your image files and click **Next** to complete the upload.
3. **Get Image URLs**:
    - After uploading the images, click on the image file to get its **Object URL**.
    - Use the URL in your web application to display the images.

---

4. Setting Up a Basic Database for Managing Movie Data

To manage the **Telugu movie data** more efficiently, you can set up a basic database such as **Amazon RDS** (for AWS) or **Cloud SQL** (for Google Cloud). This database will store the movie information (title, genre, description) in a structured way, allowing for easier management, querying, and scalability compared to a static JSON file.

In this guide, I'll explain how to set up a database using **Amazon RDS** for AWS, though similar steps can be followed for **Google Cloud SQL** if you're using GCP.

Step 1: Create an RDS Instance on AWS

1. **Go to the AWS Management Console** and navigate to the **RDS** service.
2. Click **Create Database**.
3. **Choose a Database Engine**:
    - Select **MySQL** or **PostgreSQL** (both are widely used and supported for web applications).
4. **Choose a DB Instance Type**:
    - For a basic setup, choose the free-tier eligible instance type (e.g., **db.t2.micro**).
5. **Configure Database Settings**:
    - **DB Instance Identifier**: Give your database a unique identifier (e.g., telugu-movies-db).
    - **Master Username**: Set a username for the database (e.g., admin).
    - **Master Password**: Set a secure password for the database.
6. **Storage**:
    - Configure the storage size (start with 20 GB for a basic setup).
7. **VPC and Security Group**:
    - Choose a VPC (or leave it at the default).

- Ensure the **security group** allows access from your EC2 instance (more on this in Step 3).

8. **Database Options**:
   - Set **Initial Database Name** as something like telugu_movies (this will automatically create a database within the RDS instance).
9. **Launch** the RDS instance and wait for it to be available (this may take a few minutes).

Step 2: Set Up the Movie Data Table in the RDS Database

1. **Connect to the RDS instance** from your local machine or EC2 instance:
   - You can use a tool like **MySQL Workbench**, **pgAdmin**, or directly connect via the command line:

     ```
     bash
     Copy code
     mysql -h <rds-endpoint> -u admin -p
     ```

2. **Create a Table for the Movie Data**: Once connected to the database, create a table to store movie information:

   ```
   sql
   Copy code
   CREATE TABLE movies (
       id INT AUTO_INCREMENT PRIMARY KEY,
       title VARCHAR(255) NOT NULL,
       genre VARCHAR(100),
       description TEXT
   );
   ```

3. **Insert Movie Data**: Insert your movie data into the `movies` table:

   ```
   sql
   Copy code
   INSERT INTO movies (title, genre, description) VALUES
   ('Baahubali: The Beginning', 'Action, Fantasy', 'The story of a young
   man destined for greatness in a mystical kingdom.'),
   ('Arjun Reddy', 'Drama, Romance', 'A brilliant surgeon goes on a self-
   destructive path after a failed relationship.'),
   ('Eega', 'Fantasy, Action', 'A man is reincarnated as a housefly to

   avenge his own murder.');
   ```

Step 3: Configure Security Groups for Access

1. **Security Group Configuration**:
   - Go to the **EC2 Security Groups** section.

- Edit the **security group** associated with your EC2 instance and **add an inbound rule** to allow MySQL/Aurora (port **3306**) or PostgreSQL (port **5432**), depending on your RDS setup. This ensures the EC2 instance can connect to the RDS database.

You can restrict the inbound rule to the IP address of your EC2 instance for better security.

Step 4: Modify the Web Application to Use the Database

Now that the database is set up, modify your web application to fetch movie data from the RDS database instead of the JSON file. Here's how:

1. **Install a Database Connector**: If you're using **Node.js** as your backend, install a MySQL or PostgreSQL package to interact with the database.
   - For MySQL:

     ```
     bash
     Copy code
     npm install mysql
     ```

   - For PostgreSQL:

     ```
     bash
     Copy code
     npm install pg
     ```

2. **Create a Connection to the Database**: In your `app.js` or server-side JavaScript file, establish a connection to the RDS instance:

   For MySQL:

   ```
   javascript
   Copy code
   const mysql = require('mysql');

   const connection = mysql.createConnection({
       host: '<rds-endpoint>',
       user: 'admin',
       password: 'your-password',
       database: 'telugu_movies'
   });

   connection.connect((err) => {
       if (err) throw err;
       console.log('Connected to the database!');
   });
   ```

For PostgreSQL:

```
javascript
Copy code
const { Client } = require('pg');

const client = new Client({
    host: '<rds-endpoint>',
    user: 'admin',
    password: 'your-password',
    database: 'telugu_movies'
});

client.connect((err) => {
    if (err) throw err;
    console.log('Connected to PostgreSQL database!');
});
```

3. **Query the Database**: Fetch the movie data from the database and display it in your web application:

   For MySQL:

   ```
   javascript
   Copy code
   connection.query('SELECT * FROM movies', (error, results, fields) => {
       if (error) throw error;
       console.log('Movie data:', results);
   });
   ```

   For PostgreSQL:

   ```
   javascript
   Copy code
   client.query('SELECT * FROM movies', (err, res) => {
       if (err) throw err;
       console.log(res.rows);
   });
   ```

4. **Display the Data in the Web Application**: Update your front-end code (HTML/JS) to display the movie data dynamically from the database.

5. Setting Up Auto-Scaling for Your AWS EC2 Application

Auto-scaling ensures that your appl **Setting Up Auto-Scaling for Your AWS EC2 Application**

Auto-scaling ensures that your application can handle varying traffic loads efficiently. When the traffic increases, auto-scaling automatically adds more EC2 instances to balance the load, and when the traffic decreases, it scales down to minimize costs. In this section, I'll guide you through configuring **auto-scaling** for your AWS application and document the triggers (e.g., CPU usage) that initiate scaling actions.

Steps to Set Up Auto-Scaling on AWS

Step 1: Launch an EC2 Instance

1. **Launch an EC2 Instance**:
    - Go to the **EC2 Console** and launch a new EC2 instance (Amazon Linux, Ubuntu, etc.).
    - Configure it with your application and web server (Apache, Nginx) and make sure it's working correctly.
2. **Configure Security Group**:
    - Ensure the security group has HTTP (port 80) and HTTPS (port 443) open for web traffic.

Step 2: Create a Launch Template or Configuration

A **Launch Template** specifies the EC2 instance configuration (AMI, instance type, key pair, security group, etc.) that will be used to launch new instances as part of auto-scaling.

1. **Go to the Launch Templates section** in the **EC2 Console**.
2. Click **Create Launch Template**.
3. Fill in the template details:
    - **Template Name**: Give your template a meaningful name (e.g., my-web-app-template).
    - **AMI ID**: Choose the same AMI (Amazon Machine Image) that you used for the EC2 instance you already launched.
    - **Instance Type**: Select the instance type you want (e.g., t2.micro for the free tier).
    - **Key Pair**: Use the same key pair to SSH into your instances.
    - **Security Group**: Select your pre-configured security group with ports 80 and 443 open.
4. **Create the Launch Template**.

Step 3: Create an Auto Scaling Group

The **Auto Scaling Group (ASG)** will manage the scaling of your instances based on defined policies (e.g., CPU usage). Here's how to set it up:

1. **Go to the Auto Scaling section** in the **EC2 Console**.
2. Click **Create Auto Scaling Group**.
3. **Choose the Launch Template**:
    - Select the **Launch Template** you created earlier.
4. **Configure Group Size**:
    - **Desired Capacity**: Set the starting number of instances (e.g., 1).
    - **Minimum Capacity**: Set the minimum number of instances (e.g., 1).
    - **Maximum Capacity**: Set the maximum number of instances that can be launched to handle traffic (e.g., 5).
5. **Select a Load Balancer** (Optional):
    - If you are using a **Load Balancer** (e.g., Elastic Load Balancer - ELB), you can attach it to your Auto Scaling Group to evenly distribute traffic across all instances.
6. **Configure Scaling Policies**:
    - **Step or Target Tracking Policies**:
        - **Target Tracking**: A common policy that scales instances based on average **CPU utilization**.
        - Set a target CPU utilization percentage (e.g., 70%). When the average CPU usage across instances exceeds this threshold, new instances are launched.
    - **Step Scaling**:
        - You can configure step scaling to trigger actions at multiple thresholds (e.g., scale up by 1 instance when CPU exceeds 70%, scale up by 2 when it exceeds 85%).
7. **Set Health Checks**:
    - Use **EC2 health checks** to monitor the health of the instances. If any instance fails, it will automatically be replaced by a healthy one.
8. **Review and Create the Auto Scaling Group**.

Step 4: Configure Scaling Triggers (Based on CPU Usage)

Once the **Auto Scaling Group** is created, you can configure scaling triggers to ensure your application scales up or down based on **CPU utilization** or other metrics.

1. **Go to the Auto Scaling Groups section** in the EC2 Console.
2. Select the **Auto Scaling Group** you created.
3. Click on **Automatic Scaling**.

4. Under **Scaling Policies**, choose **Create Dynamic Scaling Policy**:
    - **Scaling Policy Type**: Select **Target Tracking Scaling**.
    - **Metric Type**: Choose **Average CPU Utilization**.
    - **Target Value**: Set the desired CPU threshold (e.g., 70% CPU utilization).
    - The Auto Scaling Group will now scale the number of instances to keep the average CPU utilization near 70%. For example:
        - If CPU utilization exceeds 70%, it will add more instances.
        - If CPU utilization drops below 70%, it will terminate excess instances.

Step 5: Test the Auto Scaling Configuration

You can simulate traffic spikes to see if your auto-scaling group responds correctly:

1. **Simulate CPU Load**:
    - Log into your EC2 instance and run a command to stress the CPU:

bash

Copy code

```
sudo yum install stress -y

stress --cpu 8 --timeout 600
```

    - This command generates 100% CPU load on 8 cores for 10 minutes.
2. **Monitor Scaling Behavior**:
    - In the **Auto Scaling Console**, monitor the **Instances** section to see if new instances are launched when the CPU threshold is exceeded.
    - Once traffic or CPU usage decreases, the scaling group should automatically terminate the extra instances.

Step 6: Review and Optimize Scaling Policies

- **Cooldown Period**: Ensure you have a **cooldown period** (e.g., 300 seconds) between scaling actions to avoid over-provisioning.
- **Monitor Metrics**: Use **CloudWatch** to monitor your metrics (CPU utilization, network traffic) and adjust your scaling policies as necessary to optimize costs and performance.

Auto Scaling Configuration Summary:

- **Minimum Instances**: 1
- **Maximum Instances**: 5
- **Scaling Trigger**: Average **CPU utilization** > 70%
- **Action**: Scale out by adding 1 instance when CPU > 70%, and scale in by terminating 1 instance when CPU falls below 50%.
- **Load Balancer**: (Optional) Distributes traffic evenly across instances.
- **Health Checks**: Ensure only healthy instances are kept in the Auto Scaling Group.

ication can handle varying traffic loads efficiently. When the traffic increases, auto-scaling automatically adds more EC2 instances to balance the load, and when the traffic decreases, it scales down to minimize costs. In this section, I'll guide you through configuring **auto-scaling** for your AWS application and document the triggers (e.g., CPU usage) that initiate scaling actions.

---

5. Steps to Set Up Auto-Scaling on AWS

Step 1: Launch an EC2 Instance

1. **Launch an EC2 Instance**:
    - Go to the **EC2 Console** and launch a new EC2 instance (Amazon Linux, Ubuntu, etc.).
    - Configure it with your application and web server (Apache, Nginx) and make sure it's working correctly.
2. **Configure Security Group**:
    - Ensure the security group has HTTP (port 80) and HTTPS (port 443) open for web traffic.

Step 2: Create a Launch Template or Configuration

A **Launch Template** specifies the EC2 instance configuration (AMI, instance type, key pair, security group, etc.) that will be used to launch new instances as part of auto-scaling.

1. **Go to the Launch Templates section** in the **EC2 Console**.
2. Click **Create Launch Template**.
3. Fill in the template details:
    - **Template Name**: Give your template a meaningful name (e.g., `my-web-app-template`).
    - **AMI ID**: Choose the same AMI (Amazon Machine Image) that you used for the EC2 instance you already launched.
    - **Instance Type**: Select the instance type you want (e.g., `t2.micro` for the free tier).
    - **Key Pair**: Use the same key pair to SSH into your instances.
    - **Security Group**: Select your pre-configured security group with ports 80 and 443 open.
4. **Create the Launch Template**.

Step 3: Create an Auto Scaling Group

The **Auto Scaling Group (ASG)** will manage the scaling of your instances based on defined policies (e.g., CPU usage). Here's how to set it up:

1. **Go to the Auto Scaling section** in the **EC2 Console**.
2. Click **Create Auto Scaling Group**.
3. **Choose the Launch Template**:
    - Select the **Launch Template** you created earlier.
4. **Configure Group Size**:
    - **Desired Capacity**: Set the starting number of instances (e.g., 1).
    - **Minimum Capacity**: Set the minimum number of instances (e.g., 1).
    - **Maximum Capacity**: Set the maximum number of instances that can be launched to handle traffic (e.g., 5).
5. **Select a Load Balancer** (Optional):
    - If you are using a **Load Balancer** (e.g., Elastic Load Balancer - ELB), you can attach it to your Auto Scaling Group to evenly distribute traffic across all instances.
6. **Configure Scaling Policies**:
    - **Step or Target Tracking Policies**:
        - **Target Tracking**: A common policy that scales instances based on average **CPU utilization**.
        - Set a target CPU utilization percentage (e.g., 70%). When the average CPU usage across instances exceeds this threshold, new instances are launched.
    - **Step Scaling**:
        - You can configure step scaling to trigger actions at multiple thresholds (e.g., scale up by 1 instance when CPU exceeds 70%, scale up by 2 when it exceeds 85%).
7. **Set Health Checks**:
    - Use **EC2 health checks** to monitor the health of the instances. If any instance fails, it will automatically be replaced by a healthy one.
8. **Review and Create the Auto Scaling Group**.

Step 4: Configure Scaling Triggers (Based on CPU Usage)

Once the **Auto Scaling Group** is created, you can configure scaling triggers to ensure your application scales up or down based on **CPU utilization** or other metrics.

1. **Go to the Auto Scaling Groups section** in the EC2 Console.
2. Select the **Auto Scaling Group** you created.
3. Click on **Automatic Scaling**.
4. Under **Scaling Policies**, choose **Create Dynamic Scaling Policy**:
    - **Scaling Policy Type**: Select **Target Tracking Scaling**.
    - **Metric Type**: Choose **Average CPU Utilization**.
    - **Target Value**: Set the desired CPU threshold (e.g., 70% CPU utilization).

- The Auto Scaling Group will now scale the number of instances to keep the average CPU utilization near 70%. For example:
  - If CPU utilization exceeds 70%, it will add more instances.
  - If CPU utilization drops below 70%, it will terminate excess instances.

Step 5: Test the Auto Scaling Configuration

You can simulate traffic spikes to see if your auto-scaling group responds correctly:

1. **Simulate CPU Load**:
   - Log into your EC2 instance and run a command to stress the CPU:

     ```
     bash
     Copy code
     sudo yum install stress -y
     stress --cpu 8 --timeout 600
     ```

   - This command generates 100% CPU load on 8 cores for 10 minutes.
2. **Monitor Scaling Behavior**:
   - In the **Auto Scaling Console**, monitor the **Instances** section to see if new instances are launched when the CPU threshold is exceeded.
   - Once traffic or CPU usage decreases, the scaling group should automatically terminate the extra instances.

Step 6: Review and Optimize Scaling Policies

- **Cooldown Period**: Ensure you have a **cooldown period** (e.g., 300 seconds) between scaling actions to avoid over-provisioning.
- **Monitor Metrics**: Use **CloudWatch** to monitor your metrics (CPU utilization, network traffic) and adjust your scaling policies as necessary to optimize costs and performance.

Auto Scaling Configuration Summary:

Minimum In **Enabling HTTPS for the Application and Configuring Firewall Rules**

In this task, you will secure your web application by enabling **HTTPS** using a cloud service like **AWS Certificate Manager (ACM)** or **Azure SSL**. Additionally, you will configure **firewall rules** to restrict access to only the necessary ports, ensuring better security.

Part 1: Enabling HTTPS Using AWS Certificate Manager (ACM)

Step 1: Request an SSL Certificate Using AWS Certificate Manager (ACM)

1. **Open AWS Certificate Manager (ACM)**:
   - Go to the AWS Certificate Manager (ACM) console.
2. **Request a Certificate**:
   - Click **Request a Certificate**.
   - Select **Request a public certificate** and click **Next**.
3. **Enter Domain Names**:
   - Enter your domain name (e.g., `example.com`) that will be secured with HTTPS.
   - If you are using subdomains, you can add them as well (e.g., `www.example.com`).
4. **Choose a Validation Method**:
   - **DNS validation** (recommended): ACM will provide a DNS record (CNAME) that you need to add to your domain's DNS settings.
   - **Email validation**: ACM will send an email to the domain owner for validation.
5. **Add Tags** (Optional):
   - You can add tags to organize your certificates.
6. **Review and Confirm**:
   - Review the details and click **Confirm and Request**.
7. **Validate the Domain**:
   - For **DNS validation**, add the provided CNAME record to your domain's DNS configuration.
   - Once the DNS record is added, AWS will validate the domain ownership, and your certificate will be issued.

Step 2: Attach the SSL Certificate to an EC2 Instance or Load Balancer

Option 1: Attach SSL Certificate to an EC2 Instance (via Nginx)

1. **SSH into your EC2 Instance**:
   - Use your private key and public IP to connect to your EC2 instance.
2. **Install SSL with Nginx**:
   - If you are using **Nginx** as your web server, edit the Nginx configuration file to enable SSL.

2. **Install Certbot** (for Let's Encrypt SSL):

```
bash
Copy code
sudo apt-get update
sudo apt-get install certbot python3-certbot-nginx
```

3. **Configure Nginx for SSL**:
   - Update your Nginx configuration file to enable SSL and specify the paths to your SSL certificate and private key.

```
bash
Copy code
sudo nano /etc/nginx/sites-available/default
```

- Modify the file to include:

```
nginx
Copy code
server {
    listen 443 ssl;
    server_name example.com;

    ssl_certificate
/etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/example.com/privkey.pem;

    location / {
        proxy_pass http://localhost:3000;  # Change this as
needed
    }
}
```

4. **Obtain an SSL Certificate** (Let's Encrypt):

```
bash
Copy code
sudo certbot --nginx -d example.com
```

5. **Reload Nginx**:

```
bash
Copy code
sudo systemctl reload nginx
```

Option 2: Attach SSL Certificate to an Elastic Load Balancer (ALB)

1. **Go to EC2 Console**:
   - Navigate to **Load Balancers**.
2. **Select the Load Balancer**:
   - Choose the **Application Load Balancer (ALB)** you want to attach the certificate to.
3. **Add an HTTPS Listener**:
   - In the **Listeners** tab, click **Add Listener**.
   - Choose **HTTPS (port 443)**.
4. **Select SSL Certificate**:
   - Under the **Secure Listener Settings**, select **Choose an ACM certificate**.
   - Choose the SSL certificate you requested earlier in ACM.
5. **Security Policy**:
   - Select a security policy (you can use the default policy provided by AWS).
6. **Default Actions**:

- Set the default action (e.g., forward requests to your target group).

7. **Save Changes**:
   - The Load Balancer will now handle HTTPS traffic and forward it to your EC2 instances.

Part 2: Configuring Basic Firewall Rules

To ensure your application is secure, you need to configure firewall rules (via **AWS Security Groups**) to restrict access to only the necessary ports.

Step 1: Modify Security Group Rules

1. **Open the EC2 Console**:
   - Go to the **EC2 Dashboard** and click on **Security Groups**.
2. **Select the Security Group for Your EC2 Instance** or **Load Balancer**.
3. **Configure Inbound Rules**:
   - Add rules to only allow traffic on the following ports:
     - **Port 80 (HTTP)**: Allow this port if you still want to accept HTTP traffic.
     - **Port 443 (HTTPS)**: This is required for HTTPS traffic.
     - **Port 22 (SSH)**: Allow only from your specific IP address for secure SSH access.

   Example Inbound Rule Configuration:

Type	Protocol	Port Range	Source
HTTP	TCP	80	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
SSH	TCP	22	YOUR-IP-ADDRESS

4. **Configure Outbound Rules**:
   - Ensure outbound traffic is allowed (default is all outbound traffic allowed).
5. **Save the Changes**.

Step 2: Verify Firewall Rules

1. **Check the Firewall Configuration**:
   - Test that only ports **80** (if HTTP is enabled) and **443** (HTTPS) are accessible.
   - Use tools like **telnet** or **curl** to verify that connections on other ports (e.g., 22 for SSH) are blocked from unknown IP addresses.
2. **Access the Application**:
   - Visit `https://your-domain.com` to ensure that the application is accessible via HTTPS.

- You should see the secure padlock icon indicating the SSL certificate is working correctly.

- **stances**: 1
- **Maximum Instances**: 5
- **Scaling Trigger**: Average **CPU utilization** > 70%
- **Action**: Scale out by adding 1 instance when CPU > 70%, and scale in by terminating 1 instance when CPU falls below 50%.
- **Load Balancer**: (Optional) Distributes traffic evenly across instances.
- **Health Checks**: Ensure only healthy instances are kept in the Auto Scaling Group.

6. Enabling HTTPS for the Application and Configuring Firewall Rules

In this task, you will secure your web application by enabling **HTTPS** using a cloud service like **AWS Certificate Manager (ACM)** or **Azure SSL**. Additionally, you will configure **firewall rules** to restrict access to only the necessary ports, ensuring better security.

Part 1: Enabling HTTPS Using AWS Certificate Manager (ACM)

Step 1: Request an SSL Certificate Using AWS Certificate Manager (ACM)

1. **Open AWS Certificate Manager (ACM)**:
   - Go to the AWS Certificate Manager (ACM) console.
2. **Request a Certificate**:
   - Click **Request a Certificate**.
   - Select **Request a public certificate** and click **Next**.
3. **Enter Domain Names**:
   - Enter your domain name (e.g., example.com) that will be secured with HTTPS.
   - If you are using subdomains, you can add them as well (e.g., www.example.com).
4. **Choose a Validation Method**:
   - **DNS validation** (recommended): ACM will provide a DNS record (CNAME) that you need to add to your domain's DNS settings.
   - **Email validation**: ACM will send an email to the domain owner for validation.
5. **Add Tags** (Optional):
   - You can add tags to organize your certificates.
6. **Review and Confirm**:
   - Review the details and click **Confirm and Request**.
7. **Validate the Domain**:
   - For **DNS validation**, add the provided CNAME record to your domain's DNS configuration.
   - Once the DNS record is added, AWS will validate the domain ownership, and your certificate will be issued.

Step 2: Attach the SSL Certificate to an EC2 Instance or Load Balancer

Option 1: Attach SSL Certificate to an EC2 Instance (via Nginx)

1. **SSH into your EC2 Instance**:
   - Use your private key and public IP to connect to your EC2 instance.
2. **Install SSL with Nginx**:
   - If you are using **Nginx** as your web server, edit the Nginx configuration file to enable SSL.

2. **Install Certbot** (for Let's Encrypt SSL):

bash
Copy code
sudo apt-get update
sudo apt-get install certbot python3-certbot-nginx

3. **Configure Nginx for SSL**:
   - Update your Nginx configuration file to enable SSL and specify the paths to your SSL certificate and private key.

bash
Copy code
sudo nano /etc/nginx/sites-available/default

   - Modify the file to include:

nginx
Copy code
```
server {
    listen 443 ssl;
    server_name example.com;

    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    location / {
        proxy_pass http://localhost:3000;  # Change this as needed
    }
}
```

4. **Obtain an SSL Certificate** (Let's Encrypt):

bash
Copy code

```
sudo certbot --nginx -d example.com
```

5. **Reload Nginx**:

```
bash
Copy code
sudo systemctl reload nginx
```

Option 2: Attach SSL Certificate to an Elastic Load Balancer (ALB)

1. **Go to EC2 Console**:
   - Navigate to **Load Balancers**.
2. **Select the Load Balancer**:
   - Choose the **Application Load Balancer (ALB)** you want to attach the certificate to.
3. **Add an HTTPS Listener**:
   - In the **Listeners** tab, click **Add Listener**.
   - Choose **HTTPS (port 443)**.
4. **Select SSL Certificate**:
   - Under the **Secure Listener Settings**, select **Choose an ACM certificate**.
   - Choose the SSL certificate you requested earlier in ACM.
5. **Security Policy**:
   - Select a security policy (you can use the default policy provided by AWS).
6. **Default Actions**:
   - Set the default action (e.g., forward requests to your target group).
7. **Save Changes**:
   - The Load Balancer will now handle HTTPS traffic and forward it to your EC2 instances.

Part 2: Configuring Basic Firewall Rules

To ensure your application is secure, you need to configure firewall rules (via **AWS Security Groups**) to restrict access to only the necessary ports.

Step 1: Modify Security Group Rules

1. **Open the EC2 Console**:
   - Go to the **EC2 Dashboard** and click on **Security Groups**.
2. **Select the Security Group for Your EC2 Instance** or **Load Balancer**.
3. **Configure Inbound Rules**:
   - Add rules to only allow traffic on the following ports:
     - **Port 80 (HTTP)**: Allow this port if you still want to accept HTTP traffic.
     - **Port 443 (HTTPS)**: This is required for HTTPS traffic.
     - **Port 22 (SSH)**: Allow only from your specific IP address for secure SSH access.

Example Inbound Rule Configuration:

Type	Protocol	Port Range	Source
HTTP	TCP	80	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
SSH	TCP	22	YOUR-IP-ADDRESS

4. **Configure Outbound Rules**:
    - Ensure outbound traffic is allowed (default is all outbound traffic allowed).
5. **Save the Changes**.

Step 2: Verify Firewall Rules

1. **Check the Firewall Configuration**:
    - Test that only ports **80** (if HTTP is enabled) and **443** (HTTPS) are accessible.
    - Use tools like **telnet** or **curl** to verify that connections on other ports (e.g., 22 for SSH) are blocked from unknown IP addresses.
2. **Access the Application**:
    - Visit https://your-domain.com to ensure that the application is accessible via HTTPS.
    - You should see the secure padlock icon indicating the SSL certificate is working correctly.

Reference:

1. AWS documentation website