Web Application Code for Terraform Project

1 Application Overview

This document provides the source code for a simple Node.js web application designed to be deployed on the AWS infrastructure created using Terraform. The application serves a static HTML page with CSS styling, retrieves static assets from an S3 bucket, and connects to an RDS MySQL database to display sample data. The code is intended for deployment on EC2 instances behind an Application Load Balancer (ALB).

2 Application Files

The application consists of the following files:

- `server.js`: Node.js server to handle HTTP requests and database connections.
- `public/index.html`: Static HTML page for the web application.
- `public/styles.css`: CSS file for styling the HTML page.
- `public/sample.jpg`: Sample image to be stored in the S3 bucket.
- `package.json`: Node.js dependency configuration.

3 server.js

```
const express = require('express');
const mysql = require('mysql2');
const path = require('path');
const app = express();
const port = 3000;

// Serve static files from the 'public' directory
app.use(express.static(path.join(__dirname, 'public')));

// MySQL database connection configuration
const db = mysql.createConnection({
  host: process.env.DB_HOST || 'your-rds-endpoint', // Replace
     with RDS endpoint
  user: process.env.DB_USER || 'admin', // Replace with RDS
     username
  password: process.env.DB_PASSWORD || 'your-password', //
     Replace with RDS password
  database: process.env.DB_NAME || 'sample_db'
});

// Connect to the database
db.connect((err) => {
  if (err) {
```

```
    console.error('Database connection failed:', err.stack);
    return;
  }
  console.log('Connected to database.');
});

// Route to fetch and display users from the database
app.get('/users', (req, res) => {
  db.query('SELECT * FROM users', (err, results) => {
    if (err) {
      res.status(500).send('Error fetching users');
      return;
    }
    res.json(results);
  });
});

// Start the server
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

4 public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
       scale=1.0">
    <title>Sample Web Application</title>
    <link rel="stylesheet" href="/styles.css">
</head>
<body>
    <header>
        <h1>Welcome to Our Web Application</h1>
    </header>
    <main>
        <p>This is a sample web application deployed on AWS using
            Terraform.</p>
        <img src="https://your-s3-bucket.s3.amazonaws.com/sample.
           jpg" alt="Sample Image" width="300">
        <h2>Users from Database</h2>
        <ul id="user-list"></ul>
    </main>
    <script>
        // Fetch users from the server and display them
        fetch('/users')
            .then(response => response.json())
            .then(users => {
```

```
                const userList = document.getElementById('user-
                    list');
                users.forEach(user => {
                    const li = document.createElement('li');
                    li.textContent = `${user.name} (${user.email
                        })`;
                    userList.appendChild(li);
                });
            })
            .catch(error => console.error('Error fetching users
                :', error));
    </script>
</body>
</html>
```

5 public/styles.css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}
header {
    background-color: #333;
    color: white;
    text-align: center;
    padding: 1em;
}
main {
    max-width: 800px;
    margin: 0 auto;
    padding: 2em;
}
img {
    display: block;
    margin: 1em 0;
}
ul {
    list-style-type: none;
    padding: 0;
}
li {
    background: white;
    margin: 0.5em 0;
    padding: 1em;
    border-radius: 5px;
}
```

6 public/sample.jpg

This is a placeholder for a sample image file. Upload any image (e.g., a 300x200 pixel JPEG) to the S3 bucket specified in the Terraform infrastructure. Update the image URL in `index.html` to point to the S3 bucket (e.g., `https://your-s3-bucket.s3.amazonaws.com/sample.j`

7 package.json

```
{
  "name": "web-application",
  "version": "1.0.0",
  "description": "Sample web application for Terraform project",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.17.1",
    "mysql2": "^2.3.0"
  }
}
```

8 Setup Instructions

1. Install Node.js on the EC2 instances.
2. Clone the application code to the EC2 instances.
3. Run `npm install` in the application directory to install dependencies.
4. Set environment variables for database connection:
   - `DB_HOST`: RDS endpoint (e.g., `your-rds-endpoint.region.rds.amazonaws.com`).
   - `DB_USER`: RDS username.
   - `DB_PASSWORD`: RDS password.
   - `DB_NAME`: Database name (e.g., `sample_db`).
5. Create a `users` table in the RDS database:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL
);
INSERT INTO users (name, email) VALUES
    ('John Doe', 'john@example.com'),
    ('Jane Smith', 'jane@example.com');
```

6. Upload `sample.jpg` to the S3 bucket and update the URL in `index.html`.
7. Run `npm start` to start the server.
8. Access the application via the ALB URL (e.g., `http://your-alb-url`).

9 Deployment Notes

- Ensure the EC2 instances have the necessary IAM roles to access the S3 bucket.
- Configure security groups to allow HTTP traffic (port 80) from the ALB to EC2 and MySQL traffic (port 3306) from EC2 to RDS.
- Use the Auto Scaling Group to ensure the application remains available.
- Test the application by accessing the ALB URL and verifying that the HTML page loads, the image is served from S3, and the user list is populated from the database.